University of Technology

الجامعة التكنولوجية

Computer Science Department

قسم علوم الحاسوب

Block Cipher

التشفير الكتلي

Prof.Dr.Hala Bahjat AbdulWahab

أ.د.هاله بهجت عبدالوهاب

cs.uotechnology.edu.iq

Prof.Dr.Hala Bahjat

Symmetric Cipher Model

1. **Introduction**:

Symmetric encryption, also referred to as conventional encryption or single-key encryption, was the only type of encryption in use prior to the development of public key encryption in the 1970s. It remains by far the most widely used of the two types of encryption. We begin with a look at a general model for the symmetric encryption process; this will enable us to understand the context within which the algorithms are used. Before beginning, we define some terms. An original message is known as the **plaintext**, while the coded message is called the **ciphertext**. The process of converting from plaintext to ciphertext is known as **enciphering or encryption**; restoring the plaintext from the ciphertext is **deciphering or decryption**. The many schemes used for encryption constitute the area of study known as **cryptography**. Such a scheme is known as a **cryptographic system or a cipher**. Techniques used for deciphering a message without any knowledge of the enciphering details fall into the area of **cryptanalysis**. Cryptanalysis is what the layperson calls "breaking the code." The areas of cryptography and cryptanalysis together are called **cryptology**.

2. Symmetric Cipher Model

A symmetric encryption scheme has five ingredients (Figure 1):

1- Plaintext: This is the original intelligible message or data that is fed into the algorithm as input.

2- Encryption algorithm: The encryption algorithm performs various substitutions and transformations on the plaintext.

3-Secret key: The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time.

Prof.Dr.Hala Bahjat

The exact substitutions and transformations performed by the algorithm depend on the key.

4-Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts. The ciphertext is an apparently random stream of data and, as it stands, is unintelligible.

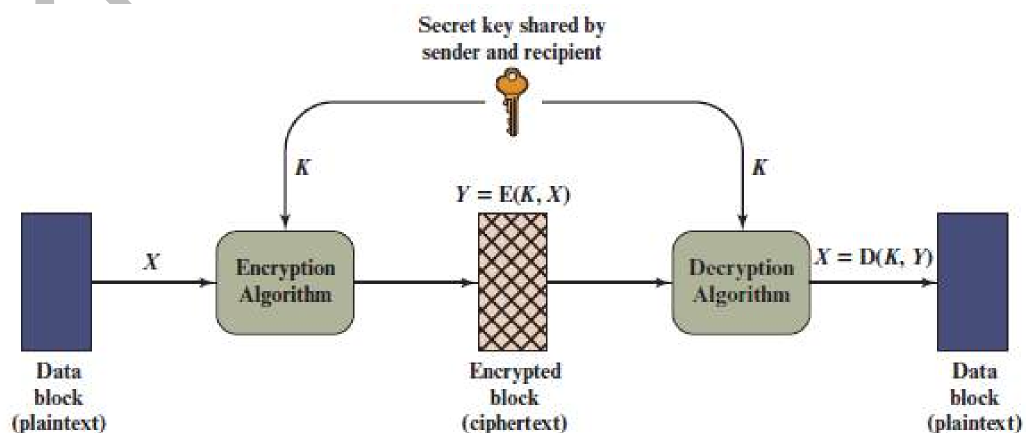
5- Decryption algorithm: This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.

2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

Prof.Dr.Hala Bahjat



Figure(1): Symmetric cipher model

2.1 Cryptographic system

Cryptographic systems are characterized along three independent dimensions:

1. The type of operations used for transforming plaintext to ciphertext : All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (i.e., that all operations are reversible). Most systems, referred to as **product systems, involve multiple stages of substitutions and transpositions.**

2. **The number of keys used**: If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.

3. **The way in which the plaintext is processed**: A block cipher processes the input one block of elements at a time, producing an output block for each input block. A stream cipher processes the input elements continuously, producing output one element at a time, as it goes along.

Prof.Dr.Hala Bahjat

3- Strength of Cryptographic Algorithms

Good cryptographic systems should always be designed so that they are as difficult to break as possible. It is possible to build systems that cannot be broken in practice (though this cannot usually be proved). This does not significantly increase system

implementation effort; however, some care and expertise is required. There is no excuse for a system designer to leave the system breakable.

Any mechanisms that can be used to circumvent security must be made explicit, documented, and brought into the attention of the end users.
In theory, any cryptographic method with a key can be broken by trying all possible keys in sequence. If using brute force to try all keys is the only option, the required computing power increases exponentially with the length of the key.

- **A 32 bit key takes 2^32 (about 10^9) steps**. This is something any amateur can do on his/her home computer.
- A **system with 56 bit keys (such as DES) takes a substantial effort**, but is quite easily breakable with special hardware.
- **Keys with 64 bits are probably breakable now by major governments**, and will be within reach of organized criminals, major companies, and lesser governments in a few years.
- **Keys with 80 bits may become breakable in future.**
- **Keys with 128 bits will probably remain unbreakable by brute force** for the foreseeable future. Even larger keys are possible; in the end we will encounter a limit where the energy consumed by the computation, using the minimum energy of a quantum mechanics operation for the energy of one step, will exceed the energy of the mass of the sun or even of the universe.
- **The key lengths used in public-key cryptography** are usually much longer than those used in symmetric ciphers. There the problem is not that of guessing the right key, but deriving the matching secret key from the public key. In the case of RSA, this is equivalent to factoring a large integer that has two large prime factors. In the case of some other cryptosystems it is equivalent to computing the discrete logarithm modulo a large integer (which is believed to be roughly comparable to factoring). Other cryptosystems are based on yet other problems.

Prof.Dr.Hala Bahjat

However, key length is not the only relevant issue. Many ciphers can be broken without trying all possible keys. In general, it is very difficult to design ciphers that could not be broken more effectively using other methods. One should generally be very wary of unpublished or secret algorithms. Quite often the designer is then not sure of the security of the algorithm, or its security depends on the secrecy of the algorithm.

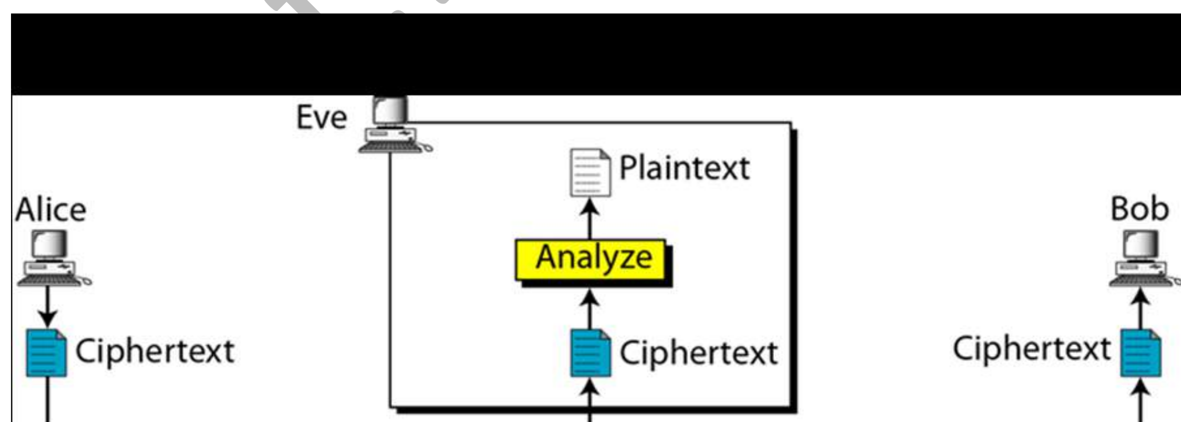
4- Cryptanalysis and Attacks on Cryptosystems

Cryptanalysis is the art of deciphering encrypted communications without knowing the proper keys. There are many cryptanalytic techniques. Some of the more important ones for a system implementer are described below.

1- **Ciphertext-only attack** (Only know algorithm / ciphertext, statistical, can identify plaintext):

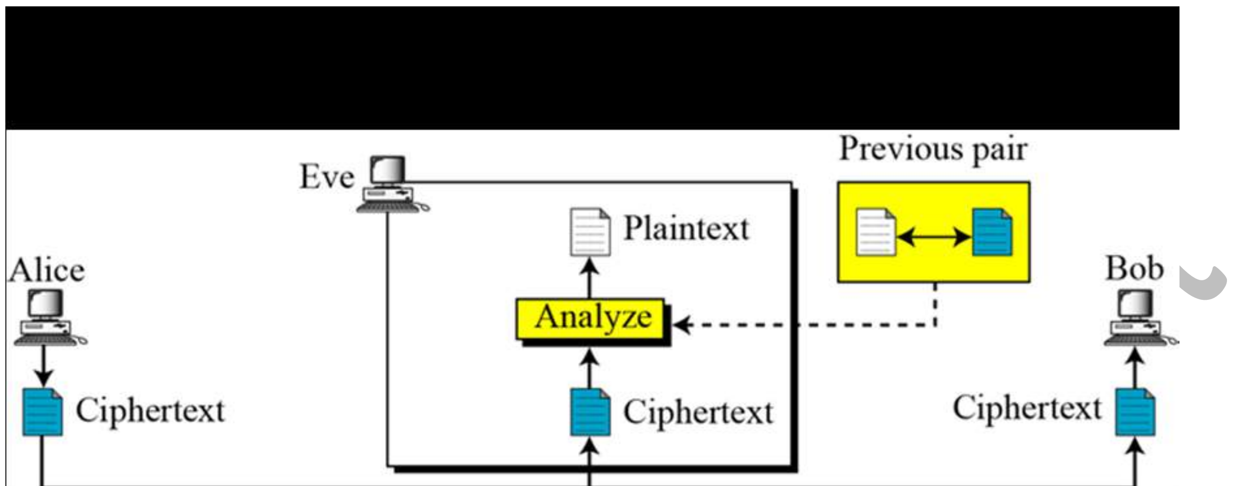
This is the situation where the attacker does not know anything about the contents of the message, and must work from ciphertext only. In practice it is quite often possible to make guesses about the plaintext, as many types of messages have fixed format headers. Even ordinary letters and documents begin in a very predictable way. It may also be possible to guess that some ciphertext block contains a common word.

f Dr.Hala Bahjat



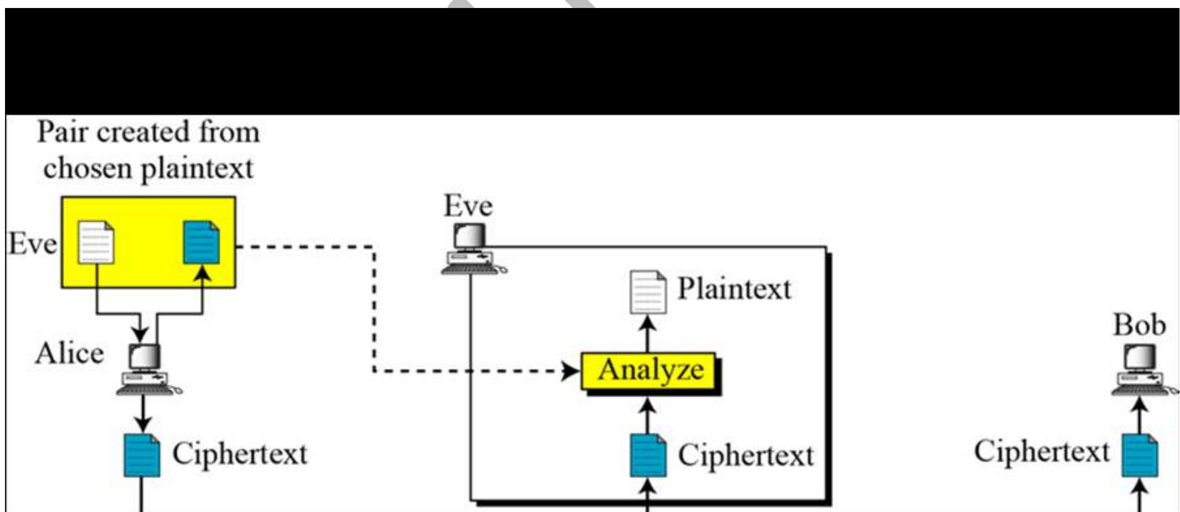
2- **Known-plaintext attack** (know/suspect plaintext & ciphertext to attack cipher):

The attacker knows or can guess the plaintext for some parts of the ciphertext. The task is to decrypt the rest of the ciphertext blocks using this information. This may be done by determining the key used to encrypt the data, or via some shortcut.

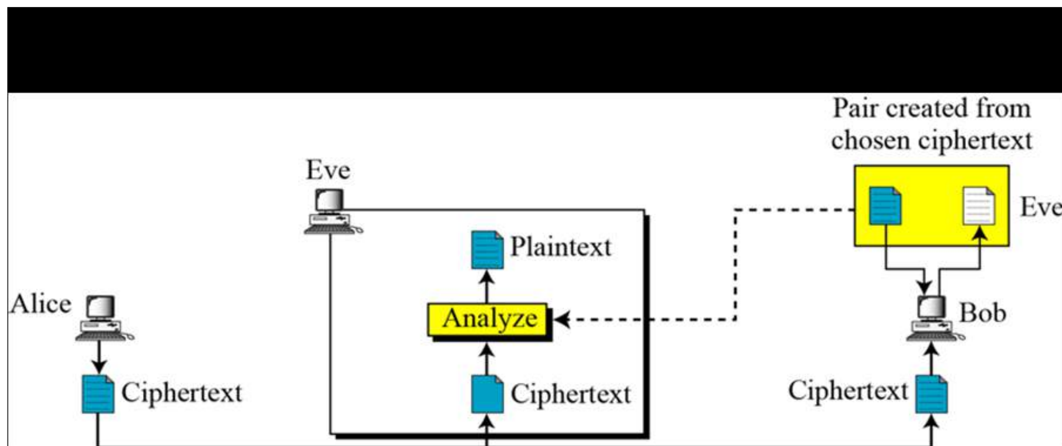


3- **Chosen-plaintext attack** (selects plaintext and obtain ciphertext to attack cipher):

The attacker is able to have any text he likes encrypted with the unknown key. The task is to determine the key used for encryption. Some encryption methods, particularly RSA, are extremely vulnerable to chosen-plaintext attacks. When such algorithms are used, extreme care must be taken to design the entire system so that an attacker can never have chosen plaintext encrypted.



4-**Chosen Ciphertext Attacks** (select ciphertext and obtain plaintext to attack cipher): Attacker obtains the decryption of any ciphertext of its choice (under the key being attacked)

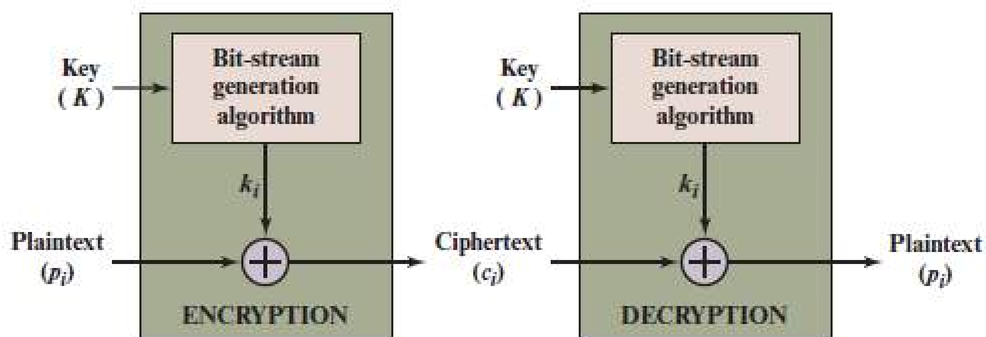


5-Stream Ciphers and Block Ciphers

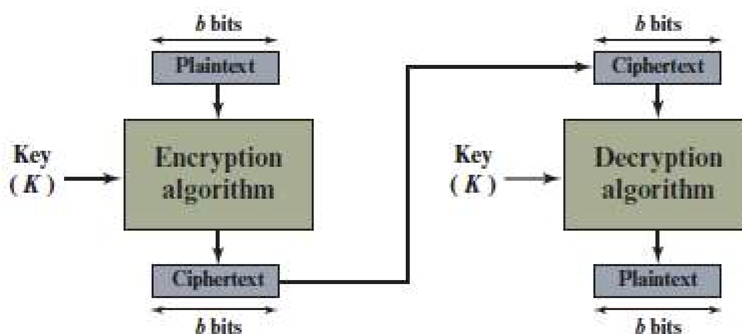
A stream cipher is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the **auto keyed ,Vigenère cipher** and the **Vernam cipher**.

A block cipher is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length.

Hala Bahjat



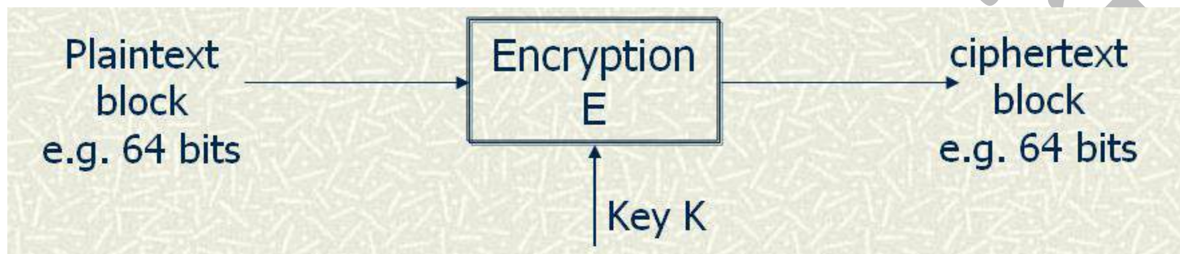
(a) Stream cipher using algorithmic bit-stream generator



(b) Block cipher

6- Block Cipher Operations

Block Cipher - An encryption scheme that "the clear text is broken up into blocks of fixed length, and encrypted one block at a time". Usually, a block cipher encrypts a block of clear text into a block of cipher text of the same length. In this case, a block cipher can be viewed as a simple substitute cipher with character size equal to the block size.



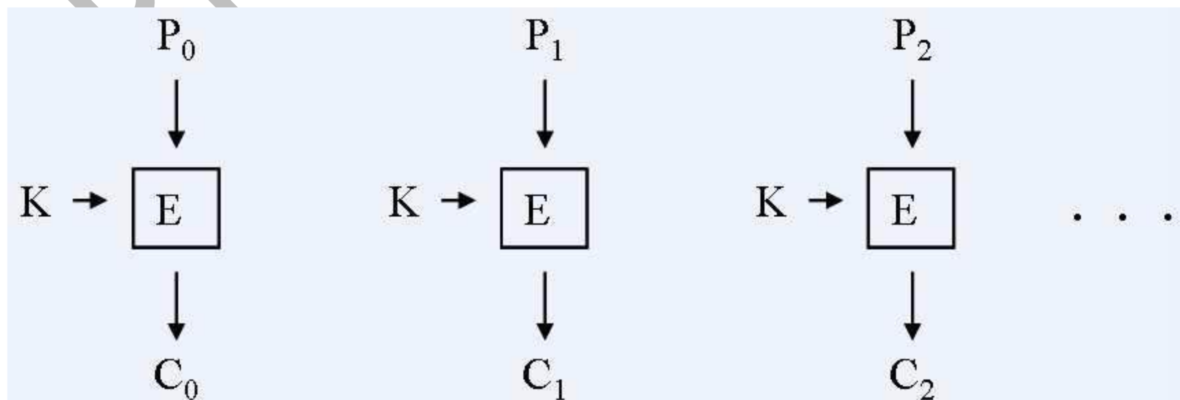
6.1 Electronic Code Book (ECB) Operation : Blocks of clear text are encrypted independently.

Main properties of (ECB) mode:

- Identical clear text blocks are encrypted to identical cipher text blocks.
- Re-ordering clear text blocks results in re-ordering cipher text blocks.
- An encryption error affects only the block where it occurs.

Electronic Code Book (ECB)*: each block encrypted separately.*

- *Encryption: Ci = Ek(Pi)*
- *Decryption: Pi = Dk(Ci)*



Prof.Dr.Hala Bahjat

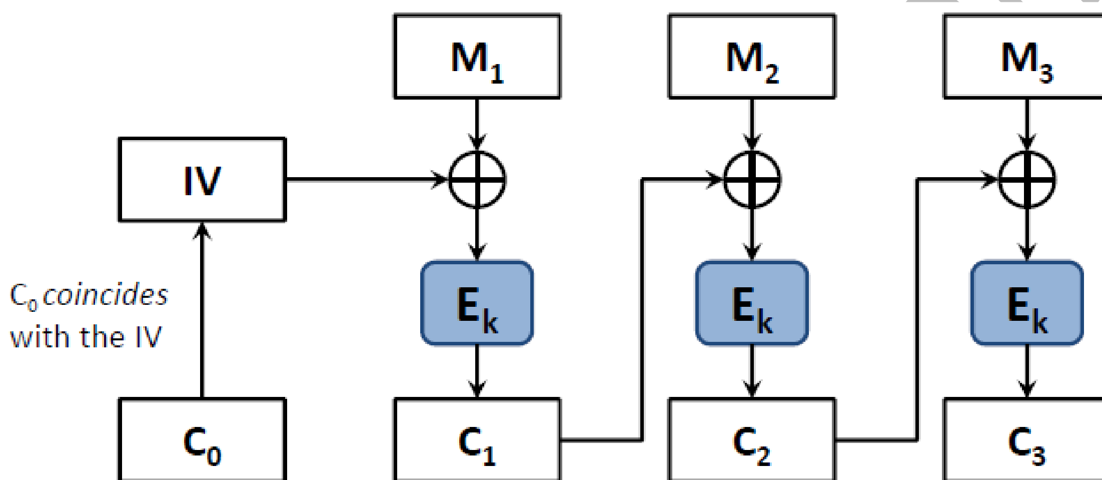
Figure : Electronic Code Book (ECB)

6.2 Cipher Block Chaining (CBC) Operation Mode - The previous cipher text block is XORed with the clear text block before applying the encryption mapping.

Main properties of this mode: An encryption error affects only the block where is occurs and one next block.

Cipher Block Chaining (CBC): *next* input depends upon *previous* output

- **Encryption**: Ci = Ek(Mi⊕Ci-1), with C0=IV
- **Decryption**: Mi = Ci-1 ⊕Dk(Ci), with C0=IV



Properties of CBC

- Randomized encryption: repeated text gets mapped to different encrypted data.
  - can be proven to be "secure" assuming that the block cipher has desirable properties and that *random IV's* are used
- A ciphertext block depends on all preceding plaintext blocks; reorder affects decryption
- Errors in one block propagate to two blocks
  - one bit error in Cj affects all bits in Mj and one bit in Mj+1
- Sequential encryption, cannot use parallel hardware

  Usage: chooses random IV and protects the integrity of IV

Prof.Dr. ...hjat

Observation:

if $C_i = C_j$ then $E_k(M_i \oplus C_{i-1}) = E_k(M_j \oplus C_{j-1})$;

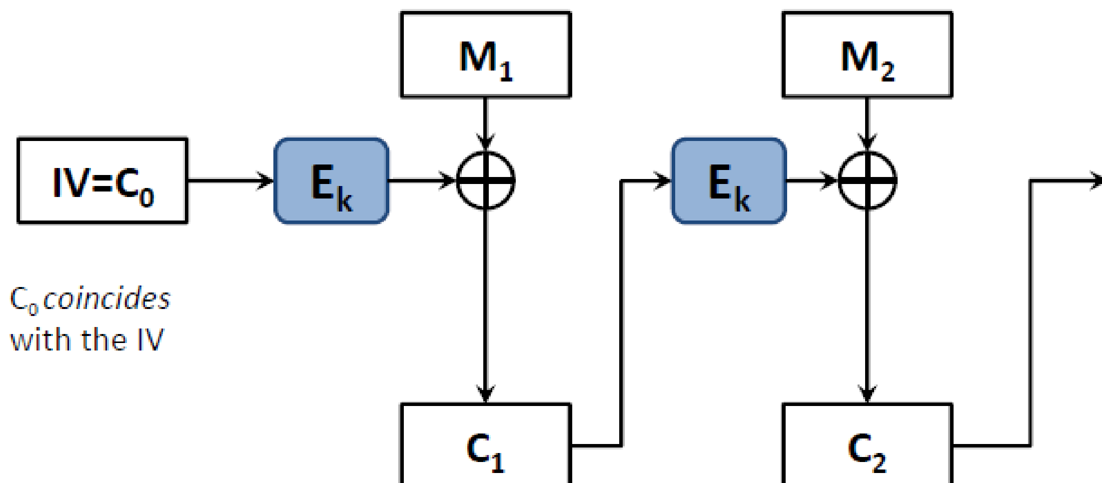
thus $M_i \oplus C_{i-1} = M_j \oplus C_{j-1}$

thus $M_i \oplus M_j = C_{i-1} \oplus C_{j-1}$

6.3 Cipher FeedBack (CFB) :Message is treated as a stream of bits , Bitwise-added to the output of the block cipher , Result is feedback for next stage (hence name)

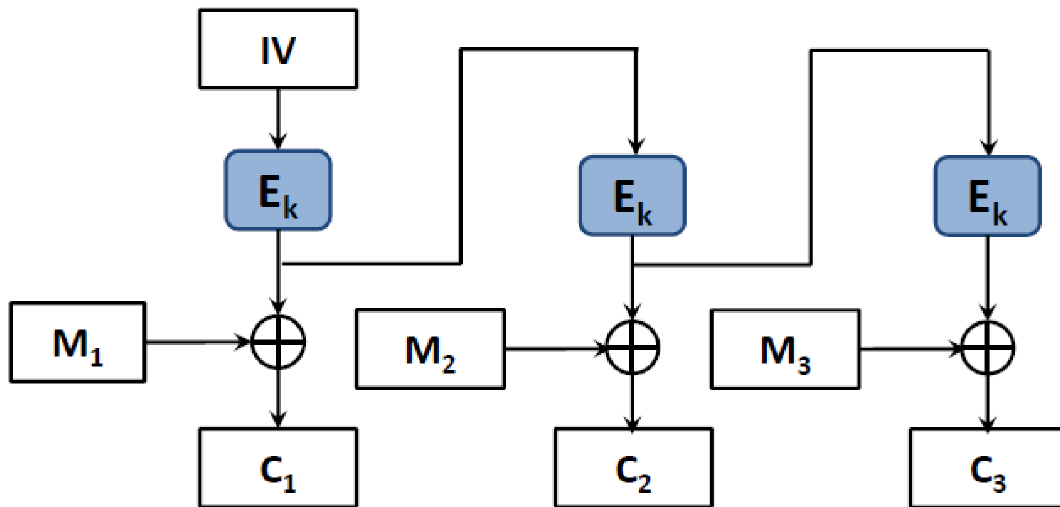
Cipher Feedback (**CFB**): the message is XORed with the feedback of encrypting the previous block

• **Encryption**: $C_i = M_i \oplus E_k(C_{i-1})$, with $C_0 = IV$



6.4 Output Feedback Mode (OFM)- The block cipher is used as a stream cipher, it produces the random key stream.

Prof.D. Sahijat



Product Cipher - An encryption scheme that "uses multiple ciphers in which the cipher text of one cipher is used as the clear text of the next cipher". Usually, substitution ciphers and transposition ciphers are used alternatively to construct a product cipher.

Iterated Block Cipher - A block cipher that "iterates a fixed number of times of another block cipher, called round function, with a different key, called round key, for each iteration".

Diffusion and Confusion (Feistel Mode)

1- Diffusion :

The terms *diffusion* and *confusion* were introduced by Claude Shannon to capture the two basic building blocks for any cryptographic system [SHAN49]. Shannon's concern was to thwart cryptanalysis based on statistical analysis. The reasoning is as follows. Assume the attacker has some knowledge of the statistical characteristics of the plaintext. For example, in a human-readable message in some language, the frequency distribution of the various letters may be known. Or there may be words or

Prof.Dr.Hala Bahjat

phrases likely to appear in the message (probable words). If these statistics are in any way reflected in the ciphertext, the cryptanalyst may be able to deduce the encryption key, part of the key, or at least a set of keys likely to contain the exact key. In what Shannon refers to as a strongly ideal cipher, all statistics of the ciphertext are independent of the particular key used.

2-Confusion: In diffusion, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext. This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally, this is equivalent to having each ciphertext digit be affected by many plaintext digits. An example of diffusion is to encrypt a message $M = m1, m2, m3$, c of characters with an averaging operation:

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

adding k successive letters to get a ciphertext letter Y_n .One can show that the statistical structure of the plaintext has been dissipated. Thus, the letter frequencies in the ciphertext will be more nearly equal than in the plaintext; the diagram frequencies will also be more nearly equal, and so on. In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation; the effect is that bits from different positions in the original plaintext contribute to a single bit of ciphertext. Every block cipher involves a transformation of a block of plaintext into a block of ciphertext, where the transformation depends on the key. The mechanism of diffusion seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible in order to thwart attempts to deduce the key. On the other hand, confusion seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, again to thwart attempts to discover the key. Thus, even if the attacker can get some handle on the statistics of the ciphertext, the way in which the key was used to produce that ciphertext is so complex as to make it difficult to

Prof.Dr.Hala Bahjat

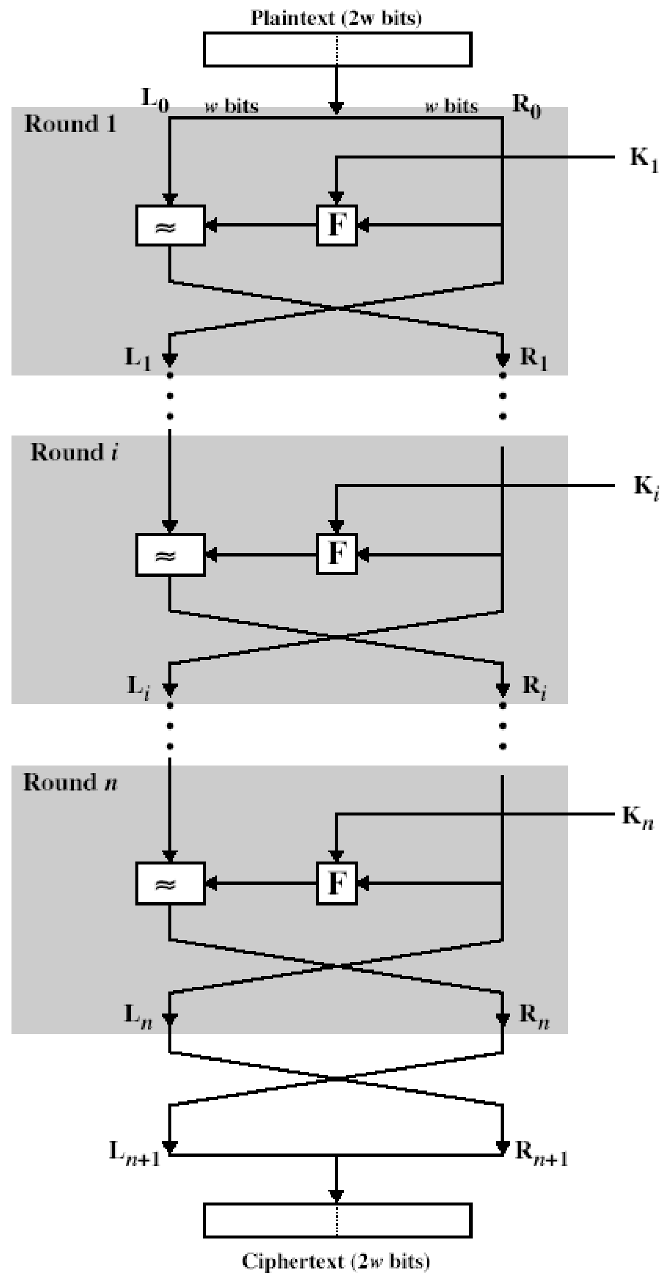
deduce the key. This is achieved by the use of a complex substitution algorithm. In contrast, a simple linear substitution function would add little confusion.

3-Feistel Mode

In cryptography, a Feistel cipher is a symmetric structure used in the construction of block ciphers, named after the German IBM cryptographer Horst Feistel; it is also commonly known as a Feistel network. A large proportion of block ciphers use the scheme, including the Data Encryption Standard (DES). The Feistel structure has the advantage that encryption and decryption operations are very similar, even identical in some cases, requiring only a reversal of the key schedule. Therefore the size of the code or circuitry required to implement such a cipher is nearly halved. Feistel networks and similar constructions are product ciphers, and so combine multiple rounds of repeated operations, such as:

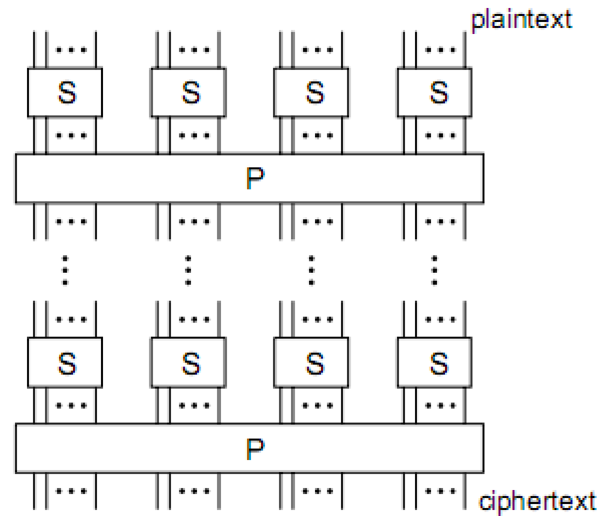
- Bit-shuffling (often called permutation boxes or P-boxes)
- Simple non-linear functions (often called substitution boxes or S-boxes)
- Linear mixing (in the sense of modular algebra) using XOR to produce a function with large amounts of what Claude Shannon described as "confusion and diffusion". Bit shuffling creates the diffusion effect, while substitution is used for confusion.

Prof.Dr.Hala Bahjat



Definition : A product cipher combines two more transformations in manner intending that the resulting cipher is more secure than the individual components.

Definition : A substitution-permutation (SP) network is a product cipher composed of a number of stages each involving substitutions and permutations .



Substitution Operation a binary word is replaced by some other binary word the whole substitution function forms the key if use n bit words, the key is 2^(n)!bits, grows rapidly

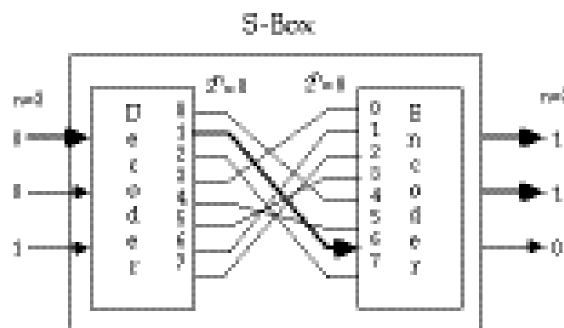
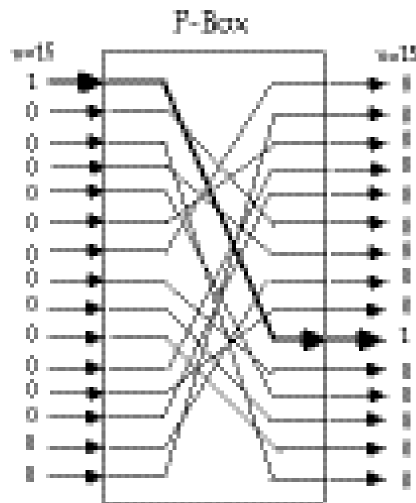


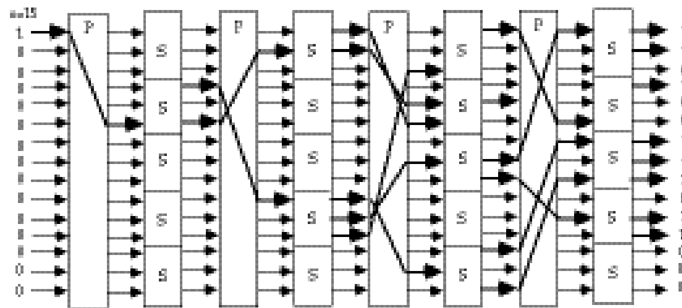
Fig 2.1 Substitution Operation

can also think of this as a large lookup table, with n address lines (hence 2^(n) addresses), each n bits wide being the output value will call them S-boxes .

Permutation Operation a binary word has its bits reordered (permuted) the re-ordering forms the key if use n bit words, the key is n!bits, which grows more slowly, and hence is less secure this is equivalent to a wire-crossing in practise (though is much harder to do in software) will call these P-boxes



Substitution-Permutation Network Shannon combined these two primitives he called these mixing transformations



• S-Boxes : Shannons mixing transformations are a special form of product ciphers where provide confusion of input bits
• P-Boxes :provide diffusion across S-box inputs in general these provide the following results

Data Encryption Standard (DES)

Data Encryption Standard (DES) : A 16-round Feistel cipher with block size of 64 bits. DES stands for Data Encryption Standard. The Data Encryption Standard (DES), known as the Data Encryption Algorithm (DEA) by ANSI and the DEA-1 by the ISO, has been most widely used block cipher in world, especially in financial industry. It encrypts 64-bit data, and uses 56-bit key with 16 48-bit sub-keys.

DES (Data Encryption Standard) history

In the early 1970s, there are no cryptagraphic equipment available on the market. Although several small companies made and sold cryptographic equipment. The equipment was all different and couldn't interoperate. No one really knew if any of it was secure; there was no independent body to certify the security.In 1972, the National Bureau of Standards (NBS), now the National Institute of Standards and Technology (NIST), initiated a program to protect computer and communications data. As part of that program, they wanted to develop a single, standard cryptographic algorithm. A single algorithm could be tested and certified, and different cryptographic equipment using it could interoperate. It would also be cheaper to implement and readily available. In the May, 1973 and again in August, 1974, the NBS issued a public request for proposals for a standard cryptographic algorithm. They specified a series of design criteria:

The algorithm must be:

- security
- completely specified
- easy to understand
- public
- available to all users
- efficient to use
- able to be validated
- exportable

Public response indicated that there was considerable interest in a cryptographic standard, but little public expertise in the field. Most of the submissions are reworked classical or machine cipher; none of the submissions came close to meeting the

Prof.Dr.Hala Bahjat

requirements. However, IBM submitted an algorithm based on one developed by IBM during the early 1970s, called Lucifer. The algorithm, although complicated, was straightforward. It used only simple logical operations on small groups of bits and could be implemented fairly efficiently in hardware. The proposed standard was adopted as a federal standard in November, 1976 and authorized for use on all unclassified government communications. ANSI approved DES as a private-sector standard in 1980.

Description of DES

DES is a block cipher; it encrypts data in 64-bit blocks. A 64-bit block of plaintext goes in one end of the algorithm and a 64-bit block of ciphertext comes out the other end. DES is a symmetric algorithm: The same algorithm and key are used for both encryption and decryption (except for minor differences in the key schedule).The key length is 56 bits. (The key is usually expressed as a 64-bit number, but every eighth bit is used for parity checking and is ignored. These parity bits are the least- significant bits of the key bytes.) The key can be any 56-bit number and can be changed at any time. All security rests within the key. At its simplest level, the algorithm is nothing more than a combination of the two basic techniques of encryption: confusion and diffusion. The fundamental building block of DES is a single combination of these techniques (a substitution followed by a permutation) on the text, based on the key. This is known as a round. DES has 16 rounds; it applies the same combination of techniques on the plaintext block 16 times (see figure 1).

Prof.Dr.Hala Bahjat

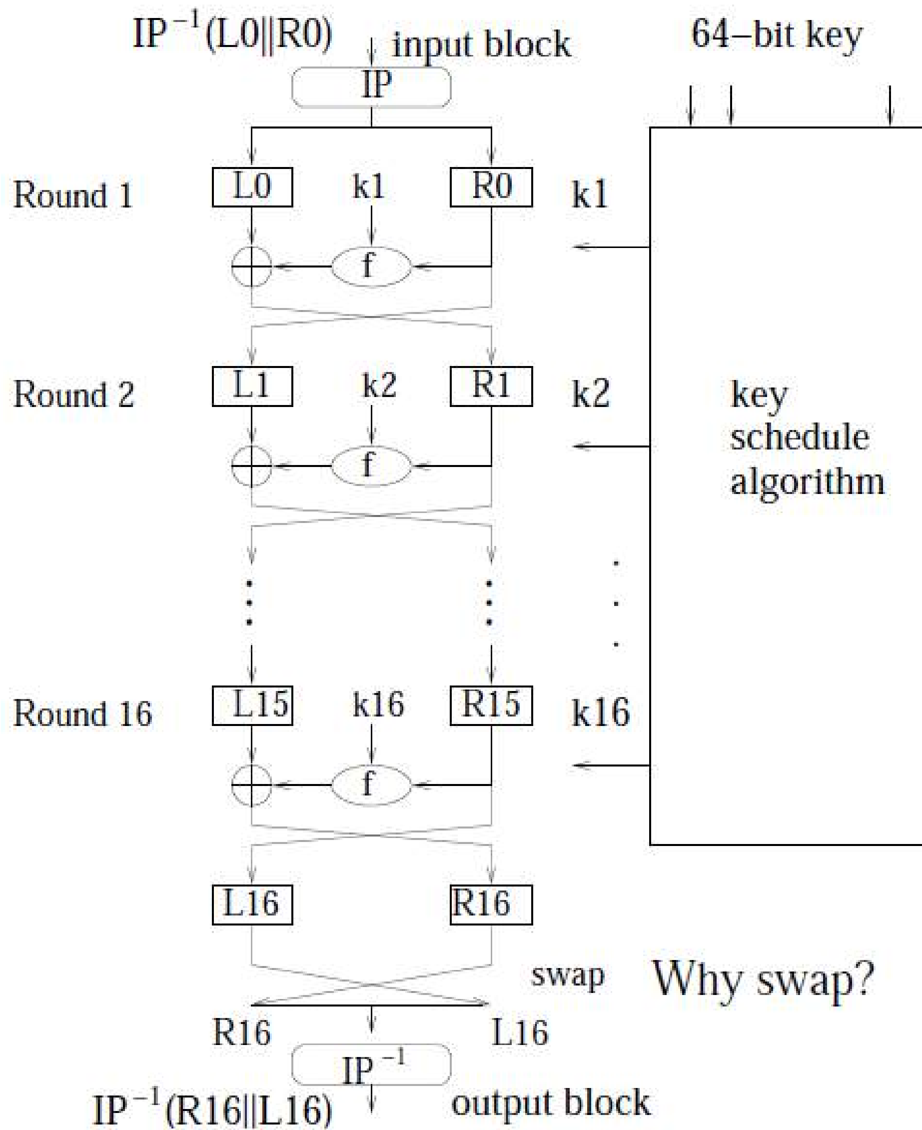


Figure 1 DES

Outline of the Algorithm

The basic process in enciphering a 64-bit data block using the DES consists of:

- an initial permutation (IP)
- 16 rounds of a complex key dependent calculation f
- final permutation, being the inverse of IP

In each round (see Figure 2,3,4,5), the key bits are shifted, and then 48 bits are selected from the 56 bits of the key. The right half of the data is expanded to 48 bits via an expansion permutation, combined with 48 bits of a shifted and permuted key via an XOR, sent through 8 S-boxes producing 32 new bits, and permuted again. These four operations make up Function f. The output of Function f is then combined with the left half via another XOR. The result of these operations becomes the new right half; the old right half becomes the new left half. If Bi is the result of the ith iteration, Li and Ri

are the left and right halves of Bi, Ki is the 48-bit key for round i, and f is the function that does all the substituting and permuting and XORing with the key, then a round looks like:

$Li = R_{j-1}$

$Ri = L_{i-1}$ Xor $f(R_{i-1}, K_i)$

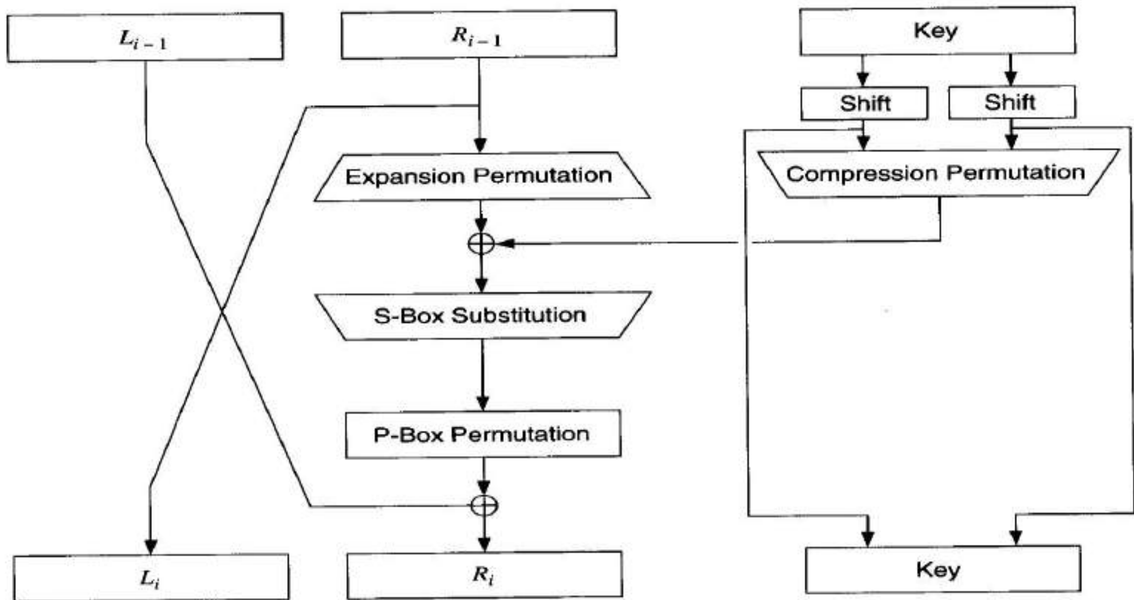


Figure 2 One round of DES

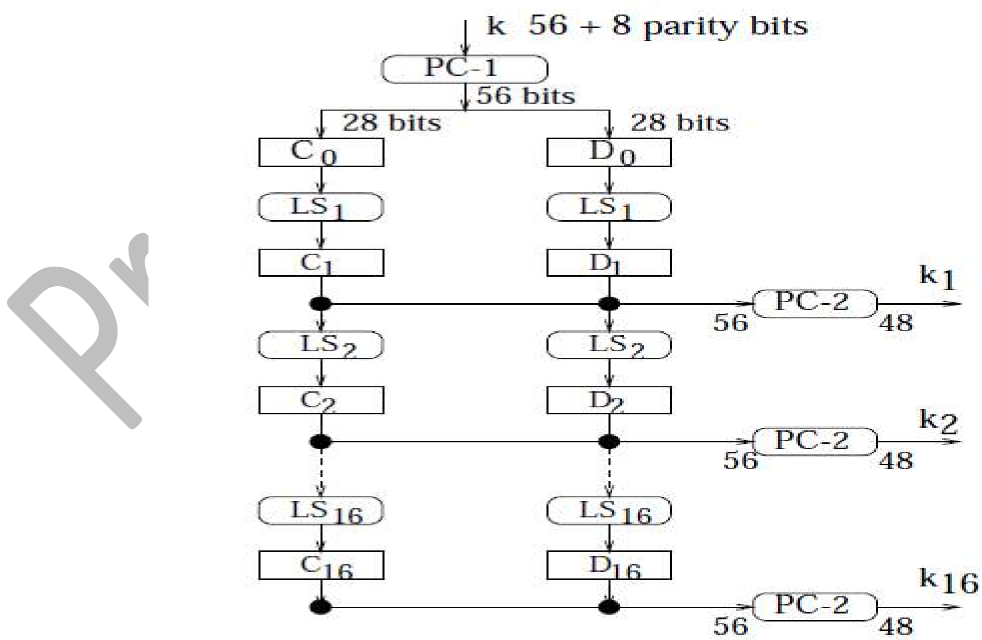


Figure 3. 16th key generation

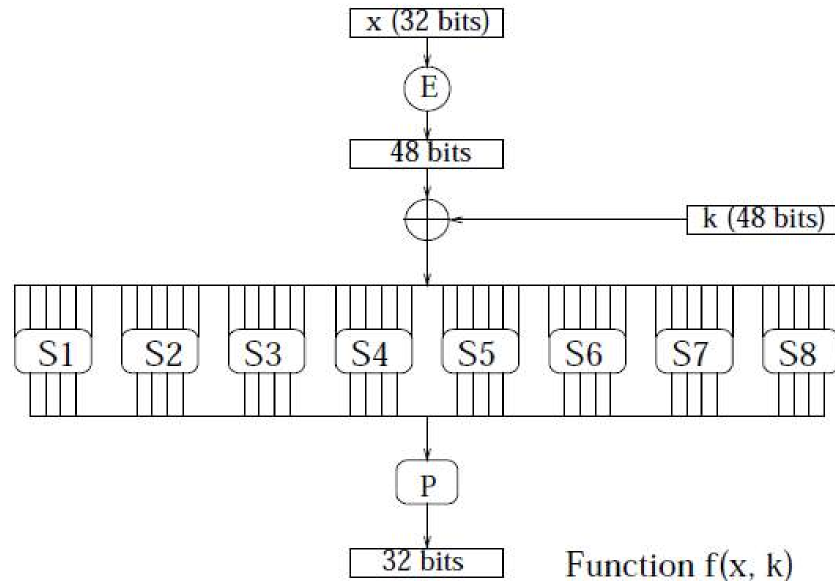


Figure 4. f-function

Prof.Dr.Hala Bahjat

The Feistel (F) Function

The F-function, depicted in Figure (4), operates on half a block (32 bits) at a time and consists of four stages:

1. ***Expansion*** **the 32-bit half-block** is expanded to 48 bits using the *expansion permutation*, denoted *E* in the diagram, by duplicating some of the bits.
2. ***Key mixing*** the result is combined with a *subkey* using an XOR operation. Sixteen 48-bit subkeys one for each round are derived from the main key using the *key schedule* (described below).
3. ***Substitution*** after mixing in the subkey, the block is divided into eight 6-bit pieces before processing by the *S-boxes*, or *substitution boxes*. Each of the eight S-boxes replaces its six input bits with four output bits according to a **non-linear transformation**, provided in the form of a look up table. The S-boxes provide the core of the security of DES without them; the cipher would be linear, and trivially breakable.
4. ***Permutation*** finally, the 32 outputs from the S-boxes are rearranged according to a fixed permutation, the *P-box*. The alternation of substitution from the S-boxes, and permutation of bits from the P-box and Expansion provides so-called

"confusion and diffusion" respectively, a concept identified by Claude Shannon in the 1940s as a necessary condition for a secure yet practical cipher.

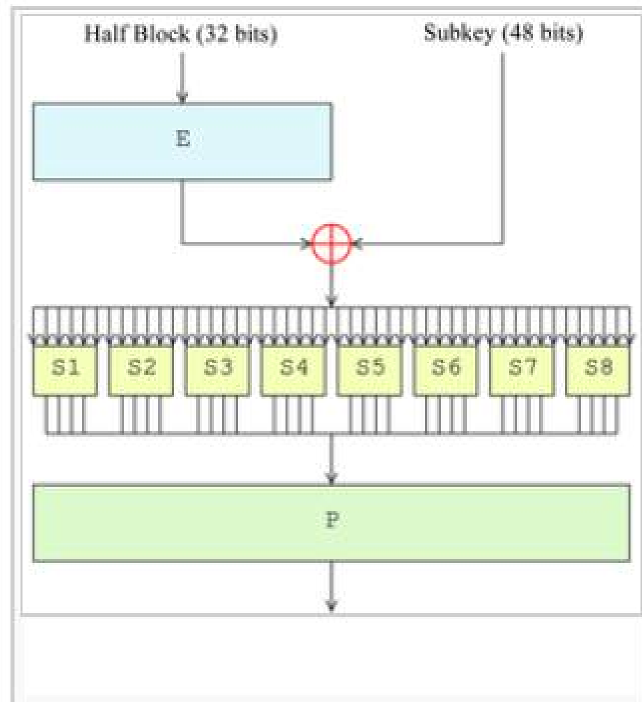
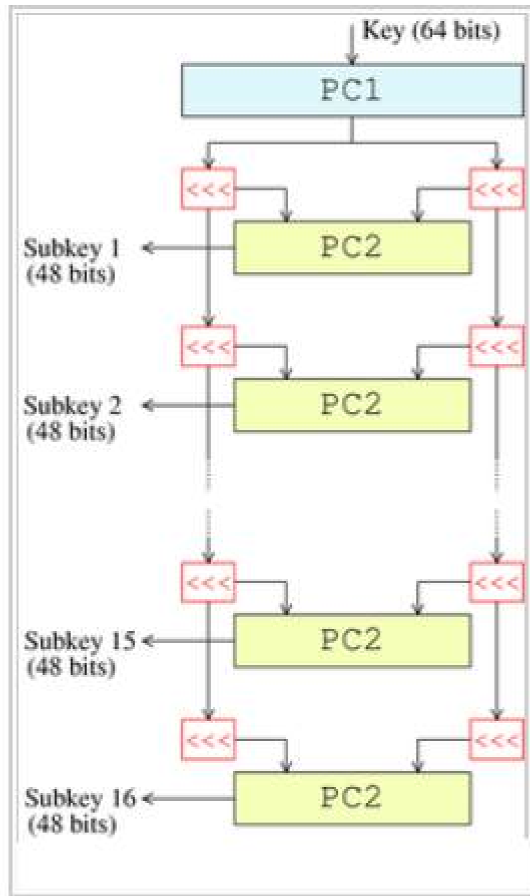


Figure (5) The Feistel Function (F-function) of DES with S-boxes

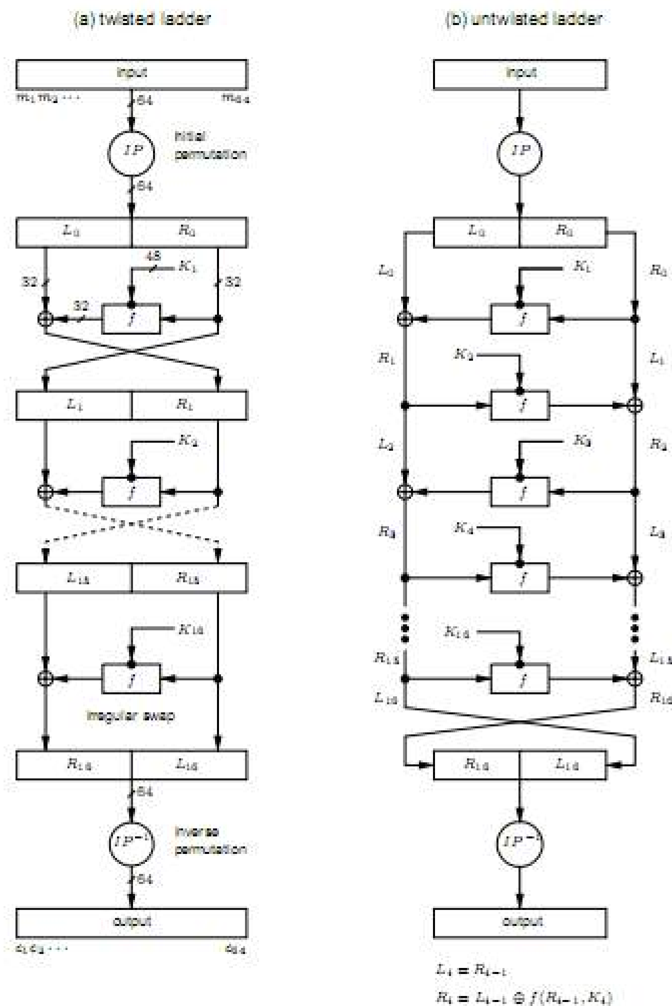
Key Schedule

Figure (6) illustrates the *key schedule* for encrypting the algorithm which generates the subkeys. Initially, 56 bits of the key are selected from the initial 64 by *Permuted Choice 1* (*PC-1*) the remaining eight bits are either discarded or used as parity check bits. The 56 bits are then divided into two 28-bit halves; each half is thereafter treated separately. In successive rounds, either halves are rotated left by one or two bits (specified for each round), and then 48 subkey bits are selected by *Permuted Choice 2* (*PC-2*) 24 bits from the left half, and 24 from the right. The rotations (denoted by "<<<" in the diagram) mean that a different set of bits is used in each subkey; The key schedule for decryption is similar the subkeys are in reverse order compared to encryption. A part from that change, the process is the same as for encryption.



Figure(6)

Prof.Dr.Ha... ahjat



Decrypting DES

After all the substitutions, permutations, XORs, and shifting around, you might think that the decryption algorithm is completely different and just as confusing as the encryption algorithm. On the contrary, the various operations were chosen to produce a very useful property: The same algorithm works for both encryption and decryption. With DES it is possible to use the same function to encrypt **or decrypt a block.**

The only difference is that the keys must be used in the reverse order. That is, if the encryption keys for each round are K1 K2 K3,..., K16 then the decryption keys are K16 K15 K14,..., K1. The algorithm that generates the key used for each round is circular as well. The key shift is a right shift and the number of positions shifted is 0,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1.

Some Preliminary Examples of DES

DES works on bits, or binary numbers--the 0s and 1s common to digital computers. Each group of four bits makes up a hexadecimal, or base 16, number. Binary "0001" is equal to the hexadecimal number "1", binary "1000" is equal to the hexadecimalnumber "8", "1001" is equal to the hexadecimal number "9", "1010" is equal to thehexadecimal number "A", and "1111" is equal to the hexadecimal number "F".

Initial Permutation (IP)

The initial permutation occurs before round 1; it transposes the input block as described in Table 1. This table, like all the other tables in this lecture, should be read left to right, top to bottom. For example, the initial permutation moves bit 58 of the plaintext to bit position 1, bit 50 to bit position 2, bit 42 to bit position 3, and so forth. The initial permutation and the corresponding final permutation do not improve DES's security, just make DES more complex.

Example: IP(675a6967 5e5a6b5a) = (ffb2194d 004df6fb)

The Key Transformation

Initially, the 64-bit DES key is reduced to a 56-bit key by ignoring every eighth bit. Let us call this operation PC1. This is described in Table 2. PC2 is the operation which reduces the 56-bits key to a 48-bits subkey for each of the 16 rounds of DES. These subkeys, Ki, are determined in the following manner. PC1 splits the key bits into 2 halves (C and D), each 28-bits. The halves C and D are circularly shifted left by either one or two bits, depending on the round. This shift is given in Table 3. After being shifted, 48 out of the 56 bits are selected. This is done by an operation called compression permutation, it permutes the order of

Example: keyinit(5b5a5767, 6a56676e) PC1(Key) C=00ffd820, D=ffec9370

Prof.Dr.Hala Bahjat

KeyRnd01 C1=01ffb040, D1=ffd926f0, PC2(C,D)=(38 09 1b 26 2f 3a 27 0f)
KeyRnd02 C2=03ff6080, D2=ffb24df0, PC2(C,D)=(28 09 19 32 1d 32 1f 2f)
KeyRnd03 C3=0ffd8200, D3=fec937f0, PC2(C,D)=(39 05 29 32 3f 2b 27 0b)
KeyRnd04 C4=3ff60800, D4=fb24dff0, PC2(C,D)=(29 2f 0d 10 19 2f 1d 3f)
KeyRnd05 C5=ffd82000, D5=ec937ff0, PC2(C,D)=(03 25 1d 13 1f 3b 37 2a)
KeyRnd06 C6=ff608030, D6=b24dfff0, PC2(C,D)=(1b 35 05 19 3b 0d 35 3b)
KeyRnd07 C7=fd8200f0, D7=c937ffe0, PC2(C,D)=(03 3c 07 09 13 3f 39 3e)
KeyRnd08 C8=f60803f0, D8=24dfffb0, PC2(C,D)=(06 34 26 1b 3f 1d 37 38)
KeyRnd09 C9=ec1007f0, D9=49bfff60, PC2(C,D)=(07 34 2a 09 37 3f 38 3c)
KeyRnd10 C10=b0401ff0, D10=26fffd90, PC2(C,D)=(06 33 26 0c 3e 15 3f 38)
KeyRnd11 C11=c1007fe0, D11=9bfff640, PC2(C,D)=(06 02 33 0d 26 1f 28 3f)
KeyRnd12 C12=0401ffb0, D12=6fffd920, PC2(C,D)=(14 16 30 2c 3d 37 3a 34)
KeyRnd13 C13=1007fec0, D13=bfff6490, PC2(C,D)=(30 0a 36 24 2e 12 2f 3f)
KeyRnd14 C14=401ffb00, D14=fffd9260, PC2(C,D)=(34 0a 38 27 2d 3f 2a 17)
KeyRnd15 C15=007fec10, D15=fff649b0, PC2(C,D)=(38 1b 18 22 1d 32 1f 37)
KeyRnd16 C16=00ffd820, D16=ffec9370, PC2(C,D)=(38 0b 08 2e 3d 2f 0e 17)

Table 1 Initial Permutation

58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7.

Table 2 Key Permutation

57,	49,	41,	33,	25,	17,	9,	1,	58,	50,	42,	34,	26,	18,
10,	2,	59,	51,	43,	35,	27,	19,	11,	3,	60,	52,	44,	36,
63,	55,	47,	39,	31,	23,	15,	7,	62,	54,	46,	38,	30,	22,
14,	6,	61,	53,	45,	37,	29,	21,	13,	5,	28,	20,	12,	4.

Number of Key Bits Shifted per Round

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 4

Compression Permutation

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,
23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,
41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,
44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32.

The Expansion Permutation

This operation expands the right half of the data, R_i, from 32 bits to 48 bits. Because this operation changes the order of the bits as well as repeating certain bits, it is known as an expansion permutation. This operation has two purposes: It makes the right half the same size as the key for the XOR operation and it provides a longer result that can be compressed during the substitution operation. However, neither of those is its main cryptographic purpose. For each 4-bit input block, the first and fourth bits each represent two bits of the output block, while the second and third bits each represent one bit of the output block. Table 5 shows which output positions correspond to which input positions. For example, the bit in position 3 of the input block moves to position 4 of the lutput block, and the bit in position 21 of the input block moves to positions 30 and 32 of the output block.

Prof.Dr.Hala

Table 5 **Expansion Permutation**

32,	1,	2,	3,	4,	5,	4,	5,	6,	7,	8,	9,
8.	9,	10,	11,	12,	13,	12,	13,	14,	15,	16,	17,
16,	17,	18,	19,	20,	21,	20,	21,	22,	23,	24,	25,
24,	25,	26,	27,	28,	29,	28,	29,	30,	31,	32,	1

The S-Box Substitution: After the compressed key is XORed with the expanded block, the 48-bit result moves to a substitution operation. The substitutions are performed by eight substitution boxes, or S-boxes. Each S-box has a 6-bit input and a 4-bit output, and there are eight different S-boxes. The 48 bits are divided into eight 6-bit sub-blocks. Each separate block is operated on by a separate S-box: The first block is operated on by S-box 1, the second block is operated on by S-box 2, and so on. Each S-box is a table of 4 rows and 16 columns. Each entry in the box is a 4-bit number. The 6 input bits of the S-box specify under which row and column number to look for the output. Table 6 shows all eight S-boxes. The input bits specify an entry in the S-box in a very particular manner. Consider an S-box input of 6 bits, labeled bi, b2, b3, b, b, and b6. Bits b, and b6are combined to form a 2-bit number, from 0 to 3, which corresponds to a row in the table. The middle 4 bits, b2 through b5, are combined to form a 4-bit number, from 0 to 15, which corresponds to a column in the table. For example, assume that the input to the sixth S-box (i.e., bits 31 through 36 of the XOR function) is 110011. The first and last bits combine to form 11, whichcorrspends to row 3 of the sixth S-box. The middle 4 bits combine to form 1001, which corresponds to the column 9 of the same S-box. The entry under row 3, column 9 of S-box 6 is 14. (Remember to count rows and columns from 0 and not from 1.) The value 1110 is substituted for 110011.The S-box substitution is the critical step in DES. The algorithm's other operators are linear and easy to analyze. The S-boxes are nonlinear and, more than any.hing else, give DES its security. The result of this substitution phase is eight 4-bit blocks which are recombined into a single 32-bit block. This block moves to the next step: the P-box permutation.

Table 6 –Boxes:

Prof.Dr.Hala Bahjat

S-box 2:

15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,

3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,

0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,

13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9,

S-box 3:

10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,

13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,

13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,

1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12,

S-box 4:

7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,

13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,

10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,

3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14,

S-box 5:

2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,

14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,

41, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,

11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3,

S-box 6:

12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,

10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,

9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,

4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13,

S-box 7:

4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,

13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,

1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,

6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12,

S-box 8:

13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,

1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,

Prof.Dr.Hala Bahjat

7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

How DES Works in Detail

DES is a ***block cipher***--meaning it operates on plaintext blocks of a given size (64-bits) and returns ciphertext blocks of the same size. Thus DES results in a ***permutation*** among the 2^64 (read this as: "2 to the 64th power") possible arrangements of 64 bits, each of which may be either 0 or 1. Each block of 64 bits is divided into two blocks of 32 bits each, a left half block **L** and a right half **R**. **(This division is only used in certain operations.)**

Example: Let **M** be the plain text message **M** = 0123456789ABCDEF, where **M** is in hexadecimal (base 16) format. Rewriting **M** in binary format, we get the 64-bit block of text:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 11011110 1111
L = 0000 0001 0010 0011 0100 0101 0110 0111
R = 1000 1001 1010 1011 1100 1101 1110 1111

The first bit of **M** is "0". The last bit is "1". We read from left to right.

DES operates on the 64-bit blocks using *key* sizes of 56- bits. The keys are actually stored as being 64 bits long, but every 8th bit in the key is not used (i.e. bits numbered 8, 16, 24, 32, 40, 48, 56, and 64). However, we will nevertheless number the bits from1 to 64, going left to right, in the following calculations. **But, as you will see, the eight bits just mentioned get eliminated when we create sub keys.**

Example: Let **K** be the hexadecimal key **K** = 133457799BBCDFF1. This gives us as the binary key (setting 1 = 0001, 3 = 0011, etc., and grouping together every eight bits, of which the last one in each group will be unused):

K = 00010011 00110100 01010111 01111001 10011011 10111100 1101111111110001

Prof.Dr.Hala Bahjat

Step 1: Create 16 sub keys, each of which is 48-bits long.

The 64-bit key is permuted according to the following table, **PC-1**. Since the first entry in the table is "57", this means that the 57th bit of the original key **K** becomes the first bit of the permuted key **K**+. The 49th bit of the original key becomes thes econd bit of the permuted key. The 4th bit of the original key is the last bit of thepermuted key. **Note only 56 bits of the original key appear in the permuted key.**

```
       PC-1
57 49 41 33 25 17 9
1 58 50 42 34 26 18
10 2 59 51 43 35 27
19 11 3 60 52 44 36
63 55 47 39 31 23 15
7 62 54 46 38 30 22
14 6 61 53 45 37 29
21 13 5 28 20 12 4
```

Example: From the original 64-bit key

K = 00010011 00110100 01010111 01111001 10011011 10111100 1101111111110001

we get the 56-bit permutation➔

K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

Next, split this key into left and right halves, ***C0*** and ***D0***, where each half has 28 bits.

Example: From the permuted key **K**+, we get

C0 = 1111000 0110011 0010101 0101111

D0 = 0101010 1011001 1001111 0001111

With ***C0*** and ***D0*** defined, we now create sixteen blocks ***Cn*** and ***Dn***, 1<=***n***<=16. Each pair of blocks ***Cn*** and ***Dn*** is formed from the previous pair ***Cn-1*** and ***Dn-1***, respectively, for ***n*** = 1, 2, ..., 16, using the following schedule of "left shifts" of the previous block**. To do a left shift, move each bit one place to the left, except for the first bit, which is cycled to the end of the block.**

Prof.Dr.Hala Bahjat

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

This means, for example, ***C3*** and ***D3*** are obtained from ***C2*** and ***D2***, respectively, by two left shifts, and ***C16*** and ***D16*** are obtained from ***C15*** and ***D15***, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that we repreviously in positions 2, 3,..., 28, 1.

Example: From original pair pair ***C0*** and ***D0*** we obtain:

C0 = 1111000011001100101010101111 ***D0*** = 0101010101100110011110001111
C1 = 1110000110011001010101011111 ***D1*** = 1010101011001100111100011110
C2 = 1100001100110010101010111111 ***D2*** = 0101010110011001111000111101
C3 = 0000110011001010101011111111 ***D3*** = 0101011001100111100011110101
C4 = 0011001100101010101111111100 ***D4*** = 0101100110011110001111010101
C5 = 1100110010101010111111110000 ***D5*** = 0110011001111000111101010101
C6 = 0011001010101011111111000011 ***D6*** = 1001100111100011110101010101
C7 = 1100101010101111111100001100 ***D7*** = 0110011110001111010101010110
C8 = 0010101010111111110000110011 ***D8*** = 1001111000111101010101011001
C9 = 0101010101111111100001100110 ***D9*** = 0011110001111010101010110011
C10 = 0101010111111110000110011001 ***D10*** = 1111000111101010101011001100
C11 = 0101011111111000011001100101 ***D11*** = 1100011110101010101100110011
C12 = 0101111111100001100110010101 ***D12*** = 0001111010101010110011001111
C13 = 0111111110000110011001010101***D 13*** = 0111101010101011001100111100
C14 = 1111111000011001100101010101 ***D14*** = 1110101010101100110011110001

Prof.Dr. Hala Bahjat

C15 = 1111100001100110010101010111 ***D15*** = 1010101010110011001111000111
C16 = 1111000011001100101010101111 ***D16*** = 0101010101100110011110001111

We now form the keys ***Kn***, for 1<=***n***<=16, by applying the following permutation table to each of the concatenated pairs ***Cn Dn***. Each pair has 56 bits, but **PC-2** only uses 48 of these.

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Therefore, the first bit of ***Kn*** is the 14th bit of ***CnDn***, the second bit the 17th, and soon, ending with the 48th bit of ***Kn*** being the 32th bit of ***CnDn***.

Example: For the first key we have ***C1D1*** = 1110000 1100110 0101010 10111111010101 0110011 0011110 0011110

which, after we apply the permutation **PC-2**, becomes

K1 = 000110 110000 001011 101111 111111 000111 000001 110010
For the other keys we have

K2 = 011110 011010 111011 011001 110110 111100 100111 100101
K3 = 010101 011111 110010 001010 010000 101100 111110 011001
K4 = 011100 101010 110111 010110 110110 110011 010100 011101
K5 = 011111 001110 110000 000111 111010 110101 001110 101000
K6 = 011000 111010 010100 111110 010100 000111 101100 101111
K7 = 111011 001000 010010 110111 111101 100001 100010 111100
K8 = 111101 111000 101000 111010 110000 010011 101111 111011
K9 = 111000 001101 101111 101011 111011 011110 011110 000001
K10 = 101100 011111 001101 000111 101110 100100 011001 001111
K11 = 001000 010101 111111 010011 110111 101101 001110 000110
K12 = 011101 010111 000111 110101 100101 000110 011111 101001
K13 = 100101 111100 010111 010001 111110 101011 101001 000001
K14 = 010111 110100 001110 110111 111100 101110 011100 111010

Prof.Dr.Hala

K15 = 101111 111001 000110 001101 001111 010011 111100 001010
K16 = 110010 110011 110110 001011 000011 100001 011111 110101

So much for the subkeys. Now we look at the message itself.

Step 2: Encode each 64-bit block of data.

There is an *initial permutation* **IP** of the 64 bits of the message data **M**. This rearranges the bits according to the following table, where the entries in the tables how the new arrangement of the bits from their initial order. The 58th bit of **M**becomes the first bit of **IP**. The 50th bit of **M** becomes the second bit of **IP**. The 7thbit of **M** is the last bit of **IP**.

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Example: Applying the initial permutation to the block of text **M**, given previously,we get

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 11011110 1111
IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 00001010 1010

Here the 58th bit of **M** is "1", which becomes the first bit of **IP**. The 50th bit of **M** is"1", which becomes the second bit of **IP**. The 7th bit of **M** is "0", which becomes thelast bit of **IP**. Next divide the permuted block **IP** into a left half ***L0*** of 32 bits, and a right half ***R0*** of32 bits.

Example: From **IP**, we get ***L0*** and ***R0***
L0 = 1100 1100 0000 0000 1100 1100 1111 1111
R0 = 1111 0000 1010 1010 1111 0000 1010 1010

Prof.Dr. ... ahjat

We now proceed through 16 iterations, for 1<=***n***<=16, using a function ***f*** which operates on two blocks--a data block of 32 bits and a key ***Kn*** of 48 bits--to produce a block of 32 bits. **Let + denote XOR addition, (bit-by-bit addition modulo 2)**. Then for **n** going from 1 to 16 we calculate

$L_n = R_{n-1} R_n = L_{n-1} + f(R_{n-1}, K_n)$

This results in a final block, for $n = 16$, of ***L16R16***. That is, in each iteration, we take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation ***f*** .

Example: For $n = 1$, we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

$L_1 = R_0$ = 1111 0000 1010 1010 1111 0000 1010 1010

$R_1 = L_0 + f(R_0, K_1)$

It remains to explain how the function f works. To calculate f, we first expand each block ***Rn-1*** from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in ***Rn-1*** . We'll call the use of this selection table the function **E**. Thus **E**(***Rn-1***) has a 32 bit input block, and a 48 bit output block.

Let **E** be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following

Prof.Dr.Hala Bahjat

E BIT-SELECTION TABLE

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

*Thus the first three bits of **E**(**Rn-1**) are the bits in positions 32, 1 and 2 of **Rn-1** while the last 2 bits of **E**(**Rn-1**) are the bits in positions 32 and 1.*

Example: We calculate **E**(***R0***) from ***R0*** as follows:

R0 = 1111 0000 1010 1010 1111 0000 1010 1010
E(***R0***) = 011110 100001 010101 010101 011110 100001 010101 010101

(**Note that each block of 4 original bits has been expanded to a block of 6 output bits.)**

Next in the ***f*** calculation, we XOR the output **E**(***Rn-1***) with the key ***Kn***:

Kn + **E**(***Rn-1***).
Example: For ***K1*** , **E**(***R0***), we have

K1 = 000110 110000 001011 101111 111111 000111 000001 110010
E(***R0***) = 011110 100001 010101 010101 011110 100001 010101 010101
K1+**E**(***R0***) = 011000 010001 011110 111010 100001 100110 010100 100111.

We have not yet finished calculating the function ***f*** . To this point we have expanded***Rn-1*** from 32 bits to 48 bits, using the selection table, and XORed the result with thekey ***Kn*** . We now have 48 bits, or eight groups of six bits. We now do somethingstrange with each group of six bits: we use them as addresses in tables called "**Sboxes**". Each group of six bits will give us an address in a different **S** box. Located atthat address will be a 4 bit number. This 4 bit number will replace the original 6 bits.The net result is that the eight groups of 6 bits are transformed into eight groups of 4bits (the 4-bit outputs from the **S** boxes) for 32 bits total.
Write the previous result, which is 48 bits, in the form:

Kn + **E**(***Rn-1***) =***B1B2B3B4B5B6B7B8***,
where each ***Bi*** is a group of six bits. We now calculate
S1(B1)S2(B2)S3(B3)S4(B4)S5(B5)S6(B6)S7(B7)S8(B8)
where ***Si(Bi)*** referres to the output of the ***i***-th **S** box.
To repeat, each of the functions ***S1, S2,..., S8***, takes a 6-bit block as input and yields a4-bit block as output. The table to determine ***S1*** is shown and explained

Prof.Dr.Hala Bahjat

S1

Row No.	Column Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

below If ***S1*** is the function defined in this table and ***B*** is a block of 6 bits, then ***S1(B)*** isdetermined as follows: The first and last bits of ***B*** represent in base 2 a number in thedecimal range 0 to 3 (or binary 00 to 11). Let that number be ***i***. The middle 4 bits of ***B***represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Letthat number be ***j***. Look up in the table the number in the ***i***-th row and ***j***-th column. It isa number in the range 0 to 15 and is uniquely represented by a 4 bit block. That blockis the output ***S1(B)*** of ***S1*** for the input ***B***. For example, for input block

B = 011011 thefirst bit is "0" and the last bit "1" giving 01 as the row. This is row 1. The middle fourbits are "1101". This is the binary equivalent of decimal 13, so the column is columnnumber 13. In row 1, column 13 appears 5. This determines the output; 5 is binary0101, so that the output is 0101. Hence ***S1***(011011) = 0101.

The tables defining the functions ***S1,...,S8*** are the following

Prof.Dr.Hala Bahjat

S1

```
14  4   13  1    2 15   11  8    3 10    6 12    5  9    0  7
 0 15    7  4   14  2   13  1   10  6   12 11    9  5    3  8
 4  1   14  8   13  6    2 11   15 12    9  7    3 10    5  0
15 12    8  2    4  9    1  7    5 11    3 14   10  0    6 13
```

S2

```
15  1    8 14    6 11    3  4    9  7    2 13   12  0    5 10
 3 13    4  7   15  2    8 14   12  0    1 10    6  9   11  5
 0 14    7 11   10  4   13  1    5  8   12  6    9  3    2 15
13  8   10  1    3 15    4  2   11  6    7 12    0  5   14  9
```

S3

```
10  0    9 14    6  3   15  5    1 13   12  7   11  4    2  8
13  7    0  9    3  4    6 10    2  8    5 14   12 11   15  1
13  6    4  9    8 15    3  0   11  1    2 12    5 10   14  7
 1 10   13  0    6  9    8  7    4 15   14  3   11  5    2 12
```

S4

```
 7 13   14  3    0  6    9 10    1  2    8  5   11 12    4 15
13  8   11  5    6 15    0  3    4  7    2 12    1 10   14  9
10  6    9  0   12 11    7 13   15  1    3 14    5  2    8  4
 3 15    0  6   10  1   13  8    9  4    5 11   12  7    2 14
```

S5

```
 2 12    4  1    7 10   11  6    8  5    3 15   13  0   14  9
14 11    2 12    4  7   13  1    5  0   15 10    3  9    8  6
 4  2    1 11   10 13    7  8   15  9   12  5    6  3    0 14
```

11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Prof.Dr.

Example: For the first round, we obtain as the output of the eight **S** boxes:

K1 + **E**(***R0***) = 011000 010001 011110 111010 100001 100110 010100 100111.

S1(B1)S2(B2)S3(B3)S4(B4)S5(B5)S6(B6)S7(B7)S8(B8) = 0101 1100 1000 0010 10110101 1001 0111

The final stage in the calculation of ***f*** is to do a permutation **P** of the **S**-box output toobtain the final value of ***f***:

f = **P**(***S1(B1)S2(B2)...S8(B8)***)

The permutation **P** is defined in the following table. **P** yields a 32-bit output from a32-bit input by

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

permuting the bits of the input block.

Example: From the output of the eight **S** boxes:

S1(B1)S2(B2)S3(B3)S4(B4)S5(B5)S6(B6)S7(B7)S8(B8) = 0101 1100 1000 0010 10110101 1001 0111
we get

f = 0010 0011 0100 1010 1010 1001 1011 1011

R1 = ***L0*** + *f*(***R0*** , ***K1***)=

1100 1100 0000 0000 1100 1100 1111 1111

+ 0010 0011 0100 1010 1010 1001 1011 1011=

1110 1111 0100 1010 0110 0101 0100 0100

In the next round, we will have ***L2*** = ***R1***, which is the block we just calculated, and then we must calculate ***R2*** =***L1*** + *f*(***R1***, ***K2***), and so on for 16 rounds. At the end of the sixteenth round we have the blocks ***L16*** and ***R16***. We then ***reverse*** the order of the two blocks into the 64-bit block

R16L16 and apply a final permutation **IP-1** as defined by the following table:

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

f.Dr.Hala Bahjat

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

Example: If we process all 16 blocks using the method defined previously, we get, onthe 16th round,

L16 = 0100 0011 0100 0010 0011 0010 0011 0100
R16 = 0000 1010 0100 1100 1101 1001 1001 0101

We reverse the order of these two blocks and apply the final permutation to

R16L16 = 00001010 01001100 11011001 10010101 01000011 01000010 0011001000110100

IP-1 = 10000101 11101000 00010011 01010100 00001111 00001010 1011010000000101

which in hexadecimal format is 85E813540F0AB405.

This is the encrypted form of **M** = 0123456789ABCDEF: namely,

C =85E813540F0AB405.

Decryption is simply the inverse of encryption, following the same steps as above, but reversing the order in which the sub keys are applied.

Prof.Dr.Hala Bahjat

CAST

CAST was designed in *Canada by Carlisle Adams and Stafford Tavares*. They claim that the name refers to their design procedure and should conjure up images of randomness, but note the authors' initials.

The example CAST algorithm uses a 64-bit block size and a 64-bit key.The structure of CAST should be familiar. The algorithm uses six S-boxes with an 8-bit input and a 32-bit output. Construction of these S-boxes is implementation-dependent and complicated. To encrypt, first divide the plaintext block into a left half and a right half. The algorithm has 8 rounds. In each round the right half is combined with some key material using function f and then XORed with **the left half to form the new right half**. The original right half (before the round) becomes the new left half. After 8 rounds (don't switch the left and right halves after the eighth round), the two halves are concatenated to form the ciphertext.

Function f is simple:

(1) Divide the 32-bit input into four 8-bit quarters: *a, b, c, d*.

(2) Divide the 16-bit sub key into two 8-bit halves: *e, f*.

(3) Process *a* through S-box 1, *b* through S-box 2, *c* through S-box 3, *d* through S-box 4, *e* through S-box 5, and *f* through S-box 6.

(4) XOR the six S-box outputs together to get the final 32-bit output.

Alternatively, the 32-bit input can be XORed with 32 bits of key, divided into four 8-bit quarters, processed through the S-boxes, and then XORed together. *N* rounds of this appears to be as secure as *N* + 2 rounds of the other option.

Prof.Dr.Hala Bahjat

Th
*k*8

Th
ne
D
de
im
de
C
th
pa
ch
en

Prof.Dr.Hala Bahjat

GOST

GOST is a block algorithm from the former Soviet Union. “GOST” is an acronym for “Gosudarstvennyi Standard,” or Government Standard, sort of similar to a FIPS, except that it can (and does) refer to just about any kind of standard. (Actually, the full name is Gosudarstvennyi Standard Soyuza SSR, or Government Standard of the Union of Soviet Socialist Republics.) This standard is number 28147-89. The Government Committee for Standards of the USSR authorized the standard, whoever they were.
I don’t know whether GOST 28147-89 was used for classified traffic or just for civilian encryption. A remark at its beginning states that the algorithm “satisfies all cryptographic requirements and not limits the grade of information to be protected.” I have heard claims that it was initially used for very high-grade communications, including classified military communications.

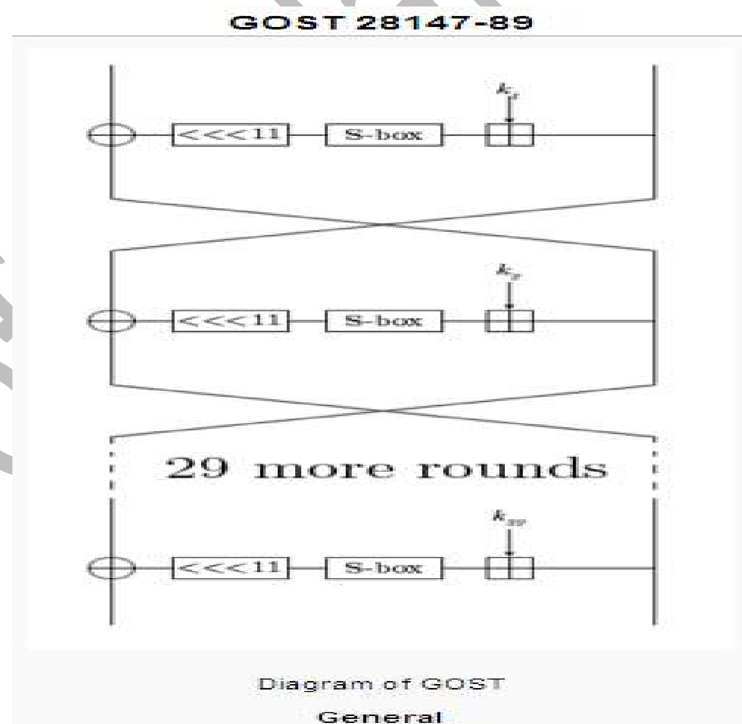


Diagram of GOST
General

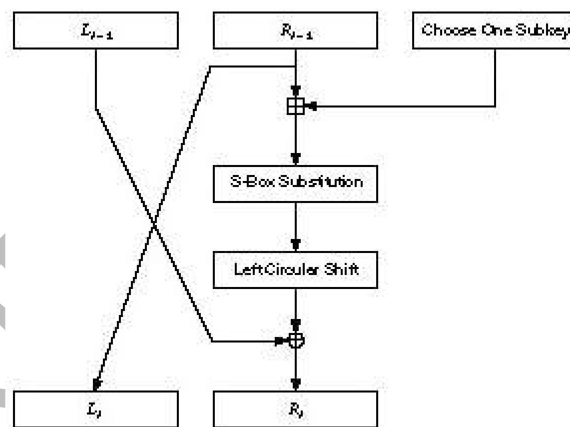
GOST is a 64-bit block algorithm with a 256-bit key. GOST also has some additional key material that will be discussed later. The algorithm iterates a simple encryption algorithm for 32 rounds.

To encrypt, first break the text up into a left half, *L*. and a right half, *R*. The subkey for round *i* is *K*i. A round, *i*, of GOST is:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

Figure is a single round of GOST. Function f is straightforward. First, the right half and the *i*th subkey are added modulo 232. The result is broken into eight 4-bit chunks, and each chunk becomes the input to a different S-box. There are eight different S-boxes in GOST; the first 4 bits go into the first S-box, the second 4 bits go into the second S-box, and so on. Each S-box is a permutation of the numbers 0 through 15. For example, an S-box might be:



7, 10, 2, 4, 15, 9, 0, 3, 6, 12, 5, 13, 1, 8, 11

In this case, if the input to the S-box is 0, the output is 7. If the input is 1, the output is 10, and so on. All eight S-boxes are different; these are considered additional key material. The S-boxes are to be kept secret.

The subkeys are generated simply. The 256-bit key is divided into eight 32-bit blocks: *k*1, *k*2,..., *k*8. Each round uses a different subkey, as shown in following

Table. Decryption is the same as encryption with the order of the *k*is reversed. The GOST standard does not discuss how to generate the S-boxes, only that they are somehow supplied. This has led to speculation that some Soviet organization would supply good S-boxes to those organizations it liked and bad S-boxes to those organizations it wished to eavesdrop on. This may very well be true, but further conversations with a GOST chip manufacturer within Russia offered.

Use of GOST Subkeys in Different Rounds

Round:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Subkey:	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Round:	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Subkey:	1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1

The outputs of the eight S-boxes are recombined into a 32-bit word, then the entire word undergoes an **11-bit left circular shift**. Finally, the result XORed to the left half to become the new right half, and the right half becomes the new left half. Do this 32 times and you're done.

The subkeys are generated simply. The 256-bit key is divided into eight 32-bit blocks: *k*1, *k*2,..., *k*8.Each round uses a different subkey, as shown in Table 14.1. Decryption is the same as encryption with the order of the *k*is reversed.
The GOST standard does not discuss how to generate the S-boxes, only that they are some how supplied [655]. This has led to speculation that some Soviet organization would supply good S-boxes to those organizations it liked and bad S-boxes to those organizations it wished to eavesdrop on. This may very well be true, but further conversations with a GOST chip manufacturer within Russia offered another alternative. He generated the S-box permutations himself, using a random-number generator.

Prof.Dr.Hala B

Cryptanalysis of GOST

These are the major differences between DES and GOST.

— DES has a complicated procedure for generating the subkeys from the keys. GOST has a very simple procedure.

— DES has a 56-bit key; GOST has a 256-bit key. If you add in the secret S-box permutations, GOST has a total of about 610 bits of secret

— The S-boxes in DES have 6-bit inputs and 4-bit outputs; the S-boxes in GOST have 4-bit inputs and outputs. Both algorithms have eight S-boxes, but an S-box in GOST is one-fourththe size of an S-box in DES.

— DES has an irregular permutation, called a P-box; GOST uses an 11-bit left circular shift.

— DES has 16 rounds; GOST has 32 rounds.

If there is no better way to break GOST other than brute force, it is a very secure algorithm. GOSThas a 256-bit key

-longer if you count the secret S-boxes. Against differential and linear cryptanalysis, GOST is probably stronger than DES.

Hala Bahjat

Table 14.2
GOST S-Boxes

S-box 1:	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
S-box 2:	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
S-box 3:	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
S-box 4:	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
S-box 5:	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
S-box 6:	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
S-box 7:	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
S-box 8:	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

Although the random S-boxes in GOST are probably weaker than the fixed S-boxes in DES, their secrecy adds to GOST's resistance against differential and linear attacks.

Also, both of these attacks depend on the number of rounds: the more rounds, the more difficult the attack. GOST has twice as many rounds as DES; this alone probably makes both differential and linear cryptanalysis infeasible.The other parts of GOST are either on par or worse than DES. GOST doesn't have the same expansion permutation that DES has. Deleting this permutation from DES weakens it by reducing the avalanche effect; it is reasonable to believe that GOST is weaker for not having it. GOST's use of addition instead is no less secure than DES's XOR.The greatest difference between them seems to be GOST's cyclic shift instead of a permutation. TheDES permutation increases the avalanche effect. In GOST a change in one input bit affects one S-box in one round, which then affects two S-boxes in the next round, three the round after that, and so on. GOST requires 8 rounds before a single change in an input affects every output bit; DES only requires 5 rounds. This is certainly a weakness. But remember: GOST has 32 rounds to DES's 16.GOST's designers tried to achieve a balance between efficiency and security. They modified DES's basic design to create an algorithm that is better suited for software implementation. They seem to have been less sure of their algorithm's security, and have tried to compensate by making the key length very large, keeping the S-boxes secret, and doubling the number of iterations. Whether their efforts have resulted in an algorithm more secure than DES remains to be seen.

Prof.Dr.Hala Bahjat

RC5

The RC5 encryption algorithm was designed by Ronald Rivest of Massachusetts Institute of Technology (MIT) and it first appeared in December 1994. RSA Data Security, Inc. estimates that RC5 and its successor, RC6, are strong candidates for potential successors to DES. RC5 analysis (RSA Laboratories) is still in progress and is periodically updated to reflect any additional findings.

Description of RC5

RC5 is a symmetric block cipher designed to be suitable for both **software** and **hardware** implementation . It is a *parameterized algorithm, with a **variable block size, a variable number of rounds** and a **variable-length key***. This provides the opportunity for great **flexibility** in both performance characteristics and the level of security.

A particular RC5 algorithm is designated as RC5-**W/R/B**. The number of bits in a word, W, is a parameter of RC5. Different choices of this parameter result in different RC5 algorithms. RC5 is iterative in structure, with a variable number of rounds. The number of rounds, **R**, is a second parameter of RC5. RC5 uses a variable-length secret key. The key length **B** (in bytes) is a third parameter of RC5. These parameters are summarized as follows:

- **W**: The word size, in bits. Allowable choice for "w" in RC5– 16, 32 and 64.. The standard value is ***32bits***; . RC5 encrypts two-word blocks so that the plaintext and ciphertext blocks are each 2w bits long. Two" word input (plaintext) block size – 64-bit plaintext "Two" word output (ciphertext) block size – 64-bit ciphertext.

- **R**: The number of rounds. Allowable values of r are 0, 1, ..., 255. Also, the expanded key table S contains t = 2 (r + 1) words.
- **B**: The number of bytes in the secret key K. Allow able values of b are 0, 1, ..., 255.
- **K**: The b-byte secret key; K[0], K[1], ...,K[b − 1].

Prof.Dr.Hala Bahjat

Example

RC5 – 32/16/10
w = 32 bits
r = 16 rounds
b = 10-byte (80-bit) secret key variable
t = 2 (r + 1) = 2 (16 + 1) = 34 words

RC5 consists of three components:

1- Key expansion algorithm.

2- Encryption algorithm.

3- Decryption algorithm.

These algorithms use the following three primitive operations:

- ⊞ Addition of words modulo 2^w
- ⊕ Bit-wise exclusive-OR of words
- <<< Rotation symbol: the rotation of **x** to the left by **y** bits is denoted by **x <<< y**.

One design feature of RC5 is its simplicity, which makes RC5 easy to implement. Another feature of RC5 is its heavy use of data-dependent rotations in encryption; this feature is very useful in preventing both differential and linear cryptanalysis.

RC5 cannot be secure for all possible values

- r = 0 (No rounds of security will provide no encryption).
- r = 1 (One round will provide very less security).
  As a matter of fact, it can be easily broken
- b = 0 (No key, no security)

Maximum allowable parameter values will be overkill.

Nominal Choice Proposed (RC5 – 32/12/16).

Key Expansion

Prof.Dr.Hala Bahjat

The key-expansion algorithm expands the user's key K to fill the expanded key table S, so that S resembles an array of $t = 2(r + 1)$ random binary words determined by K.

It uses two word-size magic constants P_w and Q_w defined for arbitrary w as shown below:

$$P_W = Odd((e - 2)2w)$$

$$Q_W = Odd((\varphi - 1)2w)$$

where

e = 2.71828 ... (base of natural logarithms)

$\varphi = (1 + \sqrt{5})/2 = 1.61803$ ... (golden ratio)

Odd(x) is the odd integer nearest to x.

Example : For w = 16 and 32 in hexadecimal form

P16 = b7e1

Q16 = 9e37

P32 = b7e15163

Q32 = 9e3779b9

First algorithmic step of key expansion: This step is to copy the secret key K[0, 1, ..., b − 1]into an array L[0, 1, ..., c − 1]of c =[b/u] words, where u = w/8 is the number of bytes/word.

This first step will be achieved by the following pseudocode operation:

for i = b – 1 down to 0 do

L[i/u] = (L[i/u] <<< 8) + K[i]

where all bytes are unsigned and the array L is initially zeroes.

Prof Dr.Hala Bahjat

Second algorithmic step of key expansion: This step is to initialize array **S** to a particular fixed pseudo-random bit pattern, using an arithmetic progression modulo **2w** determined by two constants **Pw and Qw.**

S[0] = Pw:

for i = 1 to t − 1 do

S[i] = S[i − 1] + Qw.

Third algorithmic step of key expansion: This step is to mix in the user's secret key in three passes over the arrays **S** and **L**. More precisely, due to the potentially different sizes of S and L, the larger array is processed three times, and the other array will be handled more after.

i = j = 0;

A = B = 0;

do

3∗ max (t,c) times:

A = S[i] = (S[i] + A + B) <<< 3

B = L[j] = (L[j] + A + B) <<< (A + B);

i = (i + 1)(mod t);

j = (j + 1)(mod c).

<u>Note that with the key-expansion function it is not so easy to determine K from S.</u>

Prof.Dr.Hala Bahjat

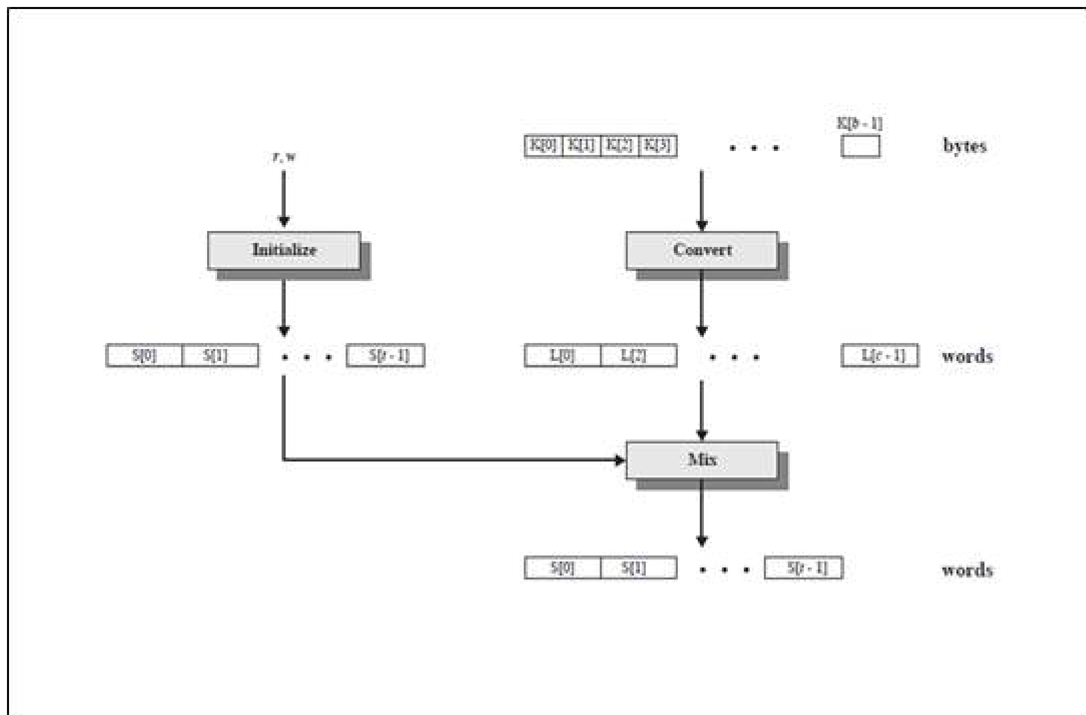


Figure (1.1) Key Expansion

<u>**Example**</u>:

Since w = 32, r = 12 and b=16

u = w/8 = 32/8 = 4 bytes/word

c =[b/u]=[16/4]= 4 words

t = 2(r + 1) = 2(12 + 1) = 26 words

The plaintext and the user's secret key are given as follows:

Plaintext = eedba521 6d8f4b15

Key = 91 5f 46 19 be 41 b2 51 63 55 a5 01 10 a9 ce 91.

1. **Key expansion**

<u>**Requirements of key expansion**</u>

Filling the expanded key table array S[0…t – 1] with random binary words

"t" – Size of table "S" => 2 (r+1)

Prof.Dr.Hala B

S table is not an “S-box” like DES. Entries in S sequentially, one at a time.

Random binary words are derived from the K.

key expansion steps:

1- Converting the Secret Key from Bytes to Words.
c =[b/u] words

where u = w/8 is the number of bytes/word.

Pseudo code for conversion:-

for i = b – 1 down to 0 do

L[i/u] = (L[i/u] <<< 8) + K[i];

where all bytes are unsigned and the array L is initially zero.

2- Initializing the S Array

Initialization to a particular fixed(key- independent)

Two magic constants

P32 = 3084996963 = 0xb7e15163

Q32 = 2654435769 = 0x9e3779b9

$S[0] = P_w$;

for i = 1 to t − 1 do

$S[i] = S[i-1] + Q_w$.

3- Mixing in the Secret Key

Pseudo code:-

```
i = j = 0;

A = B = 0;

do 3∗ max (t,c) times:

A = S[i] = (S[i] + A + B) <<< 3

B = L[j] = (L[j] + A + B) <<< (A + B);

i = (i + 1)(mod t);

j = (j + 1)(mod c);
```

Prof.Dr.Hala Bahjat

Solution

Step 1

For i = b − 1 down to 0

do L[i/u] = (L[i/u] <<< 8) + K[i] where

b = 16,u = 4 and L is initially 0.

L[i/4] = L[3] for i = 15, 14, 13 and 12.

L[3] = (L[3] <<< 8) + K[15] = 00 + 91 = 9100

L[3] = (L[3] <<< 8) + K[14] = 9100 + ce = 91ce

L[3] = (L[3] <<< 8) + K[13] = 91ce00 + a9 = 91cea9

*L[3] = (L[3] <<< 8) + K[12] = 91cea900 + 10 = 91cea910

L[i/4] = L[2] for i = 11, 10, 9 and 8.

L[2] = (L[2] <<< 8) + K[11] = 00 + 01 = 01

L[2] = (L[2] <<< 8) + K[10] = 0100 + a5 = 01a5

L[2] = (L[2] <<< 8) + K[9] = 01a500 + 55 = 01a555

*L[2] = (L[2] <<< 8) + K[8] = 01a55500 + 63 = 01a55563

L[i/4] = L[1] for i = 7, 6, 5 and 4.

L[1] = (L[1] <<< 8) + K[7] = 00 + 51 = 51

L[1] = (L[1] <<< 8) + K[6] = 5100 + b2 = 51b2

Prof.Dr.Hala Bahjat

L[1] = (L[1] <<< 8) + K[5] = 51b200 + 41 = 51b241

*L[1] = (L[1] <<< 8) + K[4] = 51b24100 + be = 51b241be

L[i/4] = L[0] for i = 3, 2, 1 and 0.

L[0] = (L[0] <<< 8) + K[3] = 00 + 19 = 19

L[0] = (L[0] <<< 8) + K[2] = 1900 + 46 = 1946

L[0] = (L[0] <<< 8) + K[1] = 194600 + 5f = 19465f

*L[0] = (L[0] <<< 8) + K[0] = 19465f00 + 91 = 19465f91

Thus, converting the secret key from bytes to words (*) yields:

L[0] = 19465f91

L[1] = 51b241be

L[2] = 01a55563

L[3] = 91cea910

Step 2

S[0] = P_{32}.

For i = 1 to 25 do

S[i] = S[i − 1] + Q_{32}:

S[0] = b7e15163

S[1] = S[0] + Q32 = b7e15163 + 9e3779b9 = 5618cb1c

S[2] = S[1] + Q32 = 5618cb1c + 9e3779b9 = f45044d5

S[3] = S[2] + Q32 = f45044d5 + 9e3779b9 = 9287be8e

Prof.Dr.Hala Bahjat

When the iterative processes continue up to $t - 1 = 2(r + 1) - 1 = 25$, we can obtain the expanded key table S as shown below:

$S[0] = b7e15163$

$S[1] = 5618cb1c$

$S[2] = f45044d5$

$S[3] = 9287be8e$

$S[4] = 30bf3847$

$S[5] = cef6b200$

$S[6] = 6d2e2bb9$

$S[7] = 0b65a572$

$S[8] = a99d1f2b$

$S[09] = 47d498e4$

$S[10] = e60c129d$

$S[11] = 84438c56$

$S[12] = 227b060f$

$S[13] = c0b27fc8$

$S[14] = 5ee9f981$

$S[15] = fd21733a$

$S[16] = 9b58ecf3$

$S[17] = 399066ac$

$S[18] = d7c7e065$

$S[19] = 75ff5a1e$

$S[20] = 1436d3d7$

$S[21] = b26e4d90$

$S[22] = 50a5c749$

$S[23] = eedd4102$.

$S[24] = 8d14babb$

$S[25] = 2b4c3474$

Step 3

$i = j = 0;\ A = B = 0;$

$3 \times \max(t, c) = 3 \times 26 = 78$ times

$A = S[i] = (S[i] + A + B) \lll 3$

$B = L[j] = (L[j] + A + B) \lll (A + B)$

$i = i + 1 (\bmod 26)$

$j = j + 1 (\bmod 4)$

$A = S[0] = (b7e15163 + 0 + 0) \lll 3$

$\quad = b7e15163 \lll 3 = bf0a8b1d$

Prof.Dr.Hala Bahjat

B = L[0] = (19465f91 + bf0a8b1d) <<< (A + B)

= d850eaae <<< bf0a8b1d = db0a1d55

A = S[1] = (5618cb1c + bf0a8b1d + db0a1d55) <<< 3

= f02d738e <<< 3 = 816b9c77

B = L[1] = (51b241be + 816b9c77 + db0a1d55) <<< (A + B)

= ae27fb8a <<< 5c75b9cc = 7fb8aae2

A = S[2] = (f45044d5 + 816b9c77 + 7fb8aae2) <<< 3

= f5748c2e <<< 3 = aba46177

B = L[2] = (01a55563 + aba46177 + 7fb8aae2) <<< (A + B)

= 2d0261bc <<< 2b5d0c59 = 785a04c3

A = S[3] = (9287be8e + aba46177 + 785a04c3) <<< 3

= b68624c8 <<< 3 = b4312645

B = L[3] = (91cea910 + b4312645 + 785a04c3) <<< (A + B)

= be59d418 <<< 2c8b2b08 = 59d418be

...

A = S[25] = (4e0d4c36 + f66a1aaf + 6d7f672f) <<< 3

= b1f6ce14,<<< 3 = 8fb670a5,

B = L[1] = (cdfc2657 + 8fb670a5 + 6d7f672f) <<< (A + B)

= cb31fe2b <<< fd35d7d4 = e2bcb31fzz

Prof.Dr.Hala Bahjat

Encryption

Encryption Algorithm

Two w-bit words are denoted as A and B

$A = A + S[0];$

$B = B + S[1];$

for $i = 1$ to r do

$A = ((A \oplus B) \lll B) + S[2 * i];$

$B = ((B \oplus A) \lll A) + S[2 * i + 1];$

The output is in the registers A and B.

Work is done on both A and B, unlike DES where only half input is updated.

Prof.Dr.Hala Bahjat

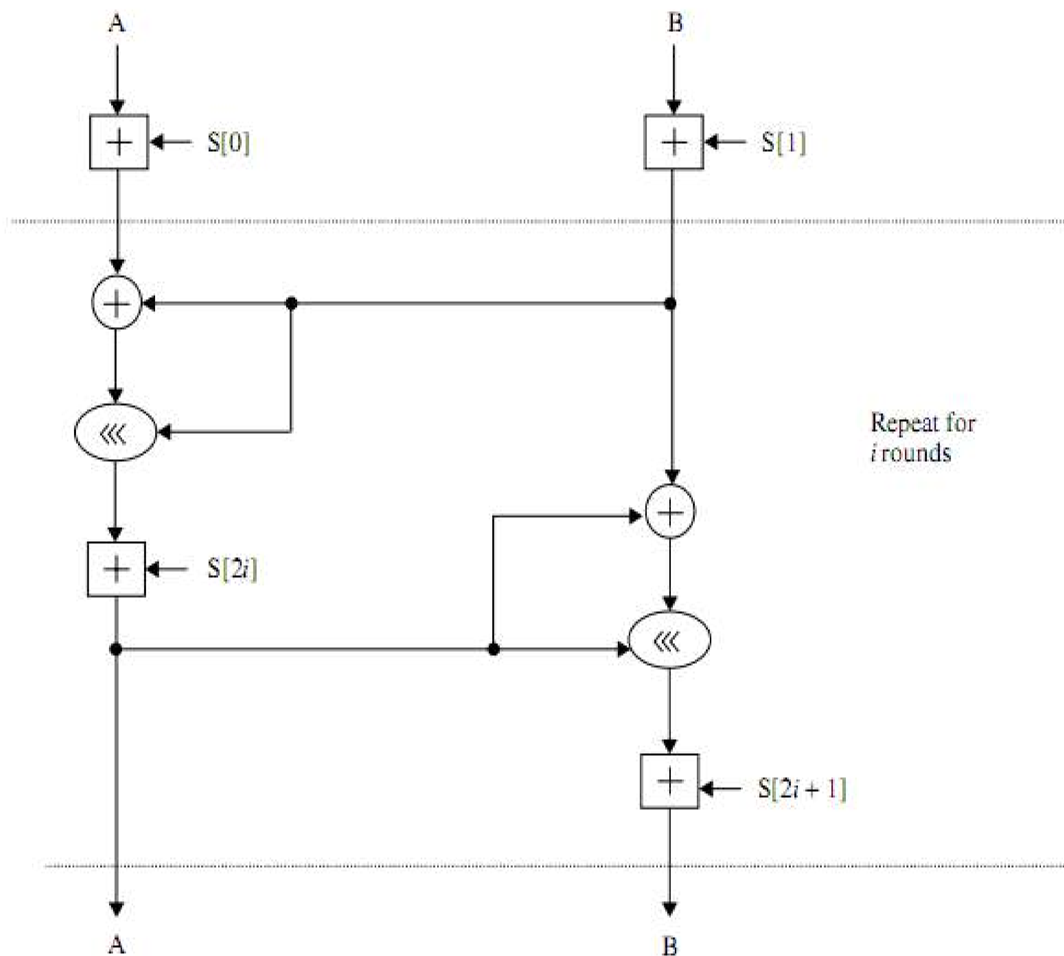


Figure (1.2) RC5 encryption algorithm.

Decryption Algorithm

- easily derived from encryption
- Two w-bit words are denoted as A and B

for i = r down to 1 do

$B = ((B - S[2 * i + 1]) \ggg A) \oplus A;$
$A = ((A - S[2 * i] \ggg B) \oplus B;$
$B = B - S[1];$
$A = A - S[0];$

The output is in the registers A and B.

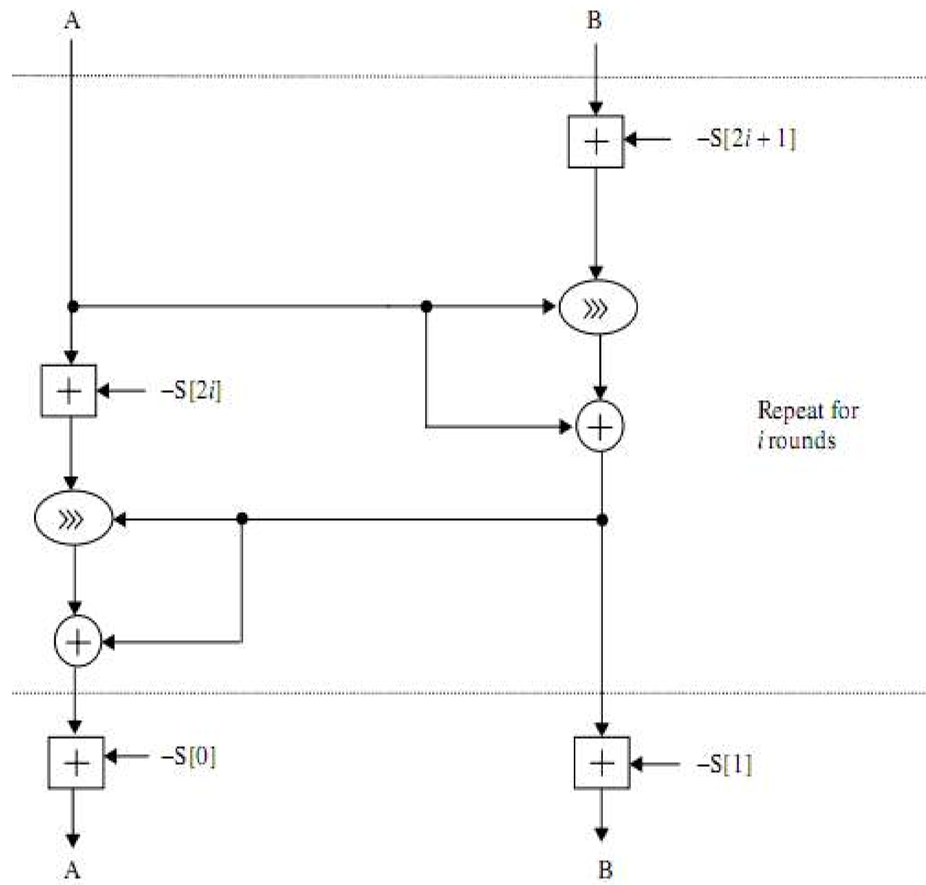


Figure (1.3) Decryption Algorithm.

Prof.Dr. ...

FEAL

FEAL (Fast data Encipherment Algorithm)was designed by Akihiro Shimizu and Shoji Miyaguchi from NTT Japan. It uses a 64-bit block and a 64-bit key. The idea was to make a DES-like algorithm with a stronger round function. Needing fewer rounds, the algorithm would run faster. Unfortunately, reality fell far short of the design goals.

Description of FEAL

In the following Figure is a block diagram of one round of FEAL. The encryption process starts with a 64-bit block of plaintext. First, the data block is XORed with 64 key bits. The data block is then split into a left half and a right half. The left half is XORed with the right half to form a new right half. The left and new right halves go through *n* rounds (four, initially). In each round the right half is combined with 16 bits of key material (using function f) and XORed with the left half to form the new right half. The original right half (before the round) forms the new left half. After *n* rounds (remember not to switch the left and right halves after the *n*th round) the left half is again XORed with the right half to form a new right half, and then the left and right halves are concatenated together to form a 64-bit whole. The data block is XORed with another 64 bits of key material, and the algorithm terminates.

Prof.Dr.Hala Bahjat

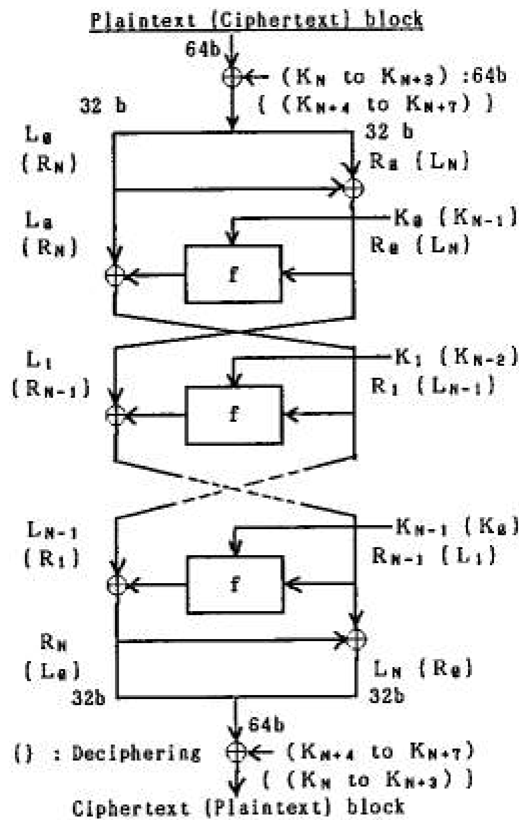


Figure:One round of FEAL.

Function f takes the 32 bits of data and 16 bits of key material and mixes them together. First the data block is broken up into 8-bit chunks, then the chunks are XORed and substituted with each other. Figure 13.4 is a block diagram of function f. The two functions $S0$ and $S1$, are defined as:

$S0(a,b)$ = rotate left two bits $((a + b) \bmod 256)$
$S1(a,b)$ = rotate left two bits $((a + b + 1) \bmod 256)$

The same algorithm can be used for decryption. The only difference is: When decrypting, the key material must be used in the reverse order.

In the following Figure is a block diagram of the key-generating function. First the 64-bit key is divided into two halves. The halves are XORed and operated on by function fk, as indicated in the diagram. Figure (fk) is a block diagram of function fk. The two 32-bit inputs are broken up into 8-bit blocks and combined and substituted as shown.

Prof.Dr.Hala Bahjat

$S0$ and $S1$ are defined as just shown. The 16-bit key blocks are then used in the encryption/decryption algorithm.

On a 10 megahertz 80286 microprocessor, an assembly-language implementation of FEAL-32 can encrypt data at a speed of 220 kilobits per second. FEAL-64 can encrypt data at a speed of 120 kilobits per second.

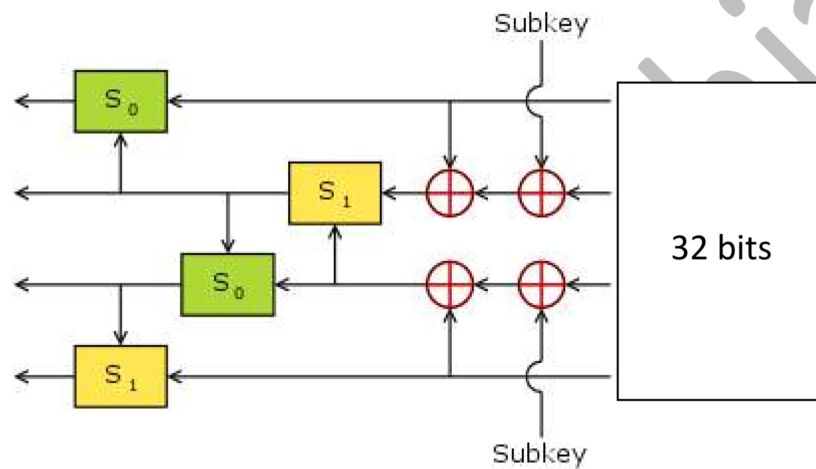


Figure : *Function f.*

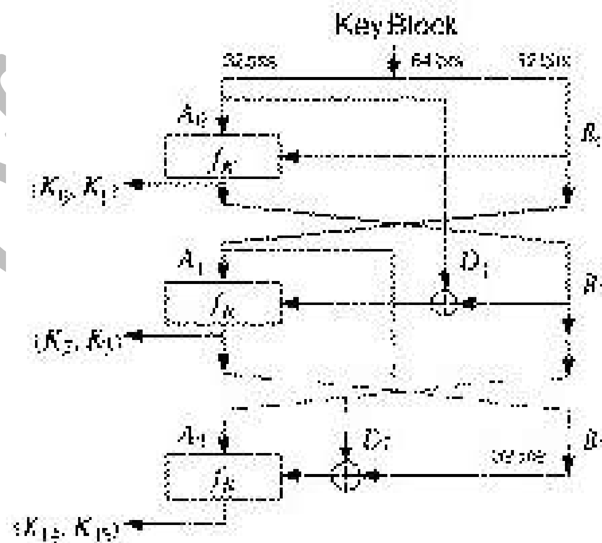


Figure : *Key processing part of FEAL.*

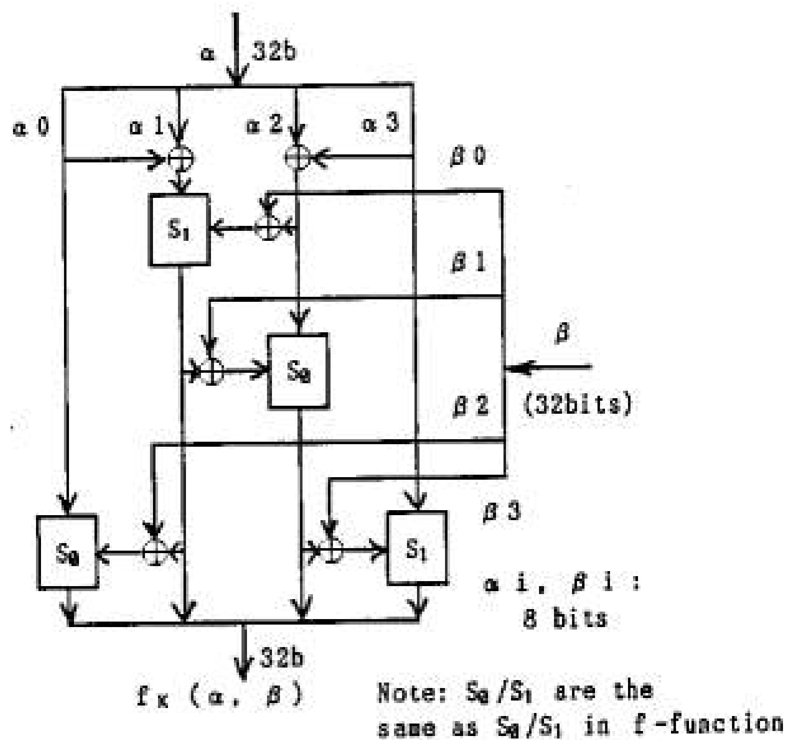


Figure :*Function fK.*

Cryptanalysis of FEAL

FEAL-4, FEAL with four rounds, was successfully cryptanalyzed with a chosen-plaintext attack in [201] and later demolished. This later attack, by Sean Murphy, was the first published differential-cryptanalysis attack and required only 20 chosen plaintexts. The designers retaliated with 8-round FEAL [1436,1437,1108] which Biham and Shamir cryptanalyzed at the SECURICOM '89 conference[1427]. Another chosen-plaintext attack, using only 10,000 blocks, against FEAL-8 [610] forced the designers to throw up their hands and define FEAL-*N* , with a variable number of rounds (greater than 8, of course).

Biham and Shamir used differential cryptanalysis against FEAL-*N*; they could break it more quickly than by brute force (with fever than 264 chosen plaintext encryptions) for *N* less than 32 . FEAL-16 required 228 chosen plaintexts or 246.5 known plaintexts to break. FEAL-8 required 2000 chosen plaintexts or 237.5 known plaintexts to break. FEAL-4 could be broken with just eight carefully selected chosen plaintexts. The FEAL designers also defined FEAL-*NX*, a modification of FEAL, that accepts 128-bit

Prof.Dr.Hala

keys. Biham and Shamir showed that FEAL-*NX* with a 128-bit key is just as easy to break as FEAL-*N* with a 64-bit. key, for any value of *N*. Recently FEAL-*N*(X)S has been proposed, which strengthens FEAL with a dynamic swapping function. There's more. Another attack against FEAL-4, requiring only 1000 known plaintexts, and against FEA°L-8, requiring only 20,000 known plaintexts. The best attack is by Mitsuru Matsui and Atshuiro Yamagishi . This is the first use of linear cryptanalysis, and can break FEAL-4 with 5 known plaintexts, FEAL-6 with 100 known plaintexts and FEAL-8 with 215 known plaintexts. Differential-linear cryptanalysis can break FEAL-8 with only 12 chosen plaintexts . Whenever someone discovers a new cryptanalytic attack, he always seems to try it out on FEAL first.

Prof.Dr.Hala Bahjat

Blow fish

1- Introduction

Blowfish was designed in 1993 by Bruce Schneier as a fast, alternative to existing encryption algorithms.it is a symmetric encryption algorithm, meaning that it uses the same secret key to both encrypt and decrypt messages. Blowfish is also a block cipher, meaning that it divides a message up into fixed length blocks during encryption and decryption**.** Blowfish is optimized for applications where the key does not change often, like a communications link or an automatic file encryptor. It is significantly faster than DES when implemented on 32-bit microprocessors with large data caches, such as the Pentium and the PowerPC.

2- Reason of appeared blow fish

- DES Weaknesses
  - S-boxes
    - Too small
    - Not sufficiently random
  - Key management / complexity

- Other Issues
  - Designed as general-purpose algorithm
  - In the public domain
    - C/C++, Java, C#, Visual Basic, Perl, Java script
  - One of the fastest block ciphers in widespread use
  - Relatively large memory footprint. Generally *not* used for:
    - Small embedded systems
    - Early smartcards

<u>Blowfish to meet the following design criteria</u>:

Prof.Dr.Hala Bahjat

- **Fast** :it encrypts data on large 32-bit microprocessors at a rate of 26 clock cycles per byte.
- **Compact**: it can run in less than 5K of memory
- **Simple**: it uses only simple operation : addition, XOR, lookup table with 32-bit operands
- **Secure**: the key length is variable, it can be as long as448 bits.

3- Description of Blowfish algorithm

Blowfish symmetric block cipher algorithm encrypts block data of 64-bits at a time.it will follows the feistel network .

Feistel Network is the fundamental principle that is based on splitting up the block of N bits in two halfs, each of size N/2(N must be even).

this algorithm is divided into two parts: Key expansion and Data encryption.

1. Key-expansion:

It will converts a key of at most 448 bits into several subkey arrays totaling 4168 bytes.

2. Data-encryption :

It is having a function to iterate 16 times of network. Each round consists of a key-dependent permutation, and a key- and data- dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookups for each round.

Prof.Dr.Hala Bahjat

Key generation:

- Blowfish uses a large number of subkeys. These keys must be precomputed before any data encryption or decryption.

- The p-array consists of 18 32-bit subkeys:

P1,P2,…………,P18

- Four 32-bit S-Boxes consists of 256 entries each:

S1,0 , S1,1,………. S1,255

S2,0 , S2,1,……….. S2,255

S3,0 , S3,1,……….. S3,255

S4,0 , S4,1,………... S4,255

Steps to generate subkeys:

1). Initialize first the P-array and then the four S-boxes, in order, with a fixed string. This string consists of the hexadecimal digits of p.

For example:

P1 = 0x243f6a88

P2= 0x85a308d3

P3 = 0x13198a2e

P4 = 0x03707344

2). XOR P1 with the first 32 bits of the key, XOR P2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P18). Repeatedly cycle through the key bits until the entire P-array has been XORed with key bits.

3). Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps (1) and (2).

4). Replace P1 and P2 with the output of step (3).

Prof.Dr.Hala Bahjat

5). Encrypt the output of step (3) using the Blowfish algorithm with the modified subkeys.

6). Replace P3 and P4 with the output of step (5).

7). Continue the process, replacing all entries of the P-array, and then all four S-boxes in order, with the output of the continuously-changing Blowfish algorithm.

In total, 521 iterations are required to generate all required subkeys. Applications can store the subkeys rather than execute this derivation process multiple times.

4- Data Encryption:

Blowfish algorithm:

Blowfish is a Feistel network consisting of 16 rounds. The input is a 64-bit data element, X . to encrypt:

Divide x into two 32-bit halves: X_L, X_R

For i = 1 to 16 :

$X_L = X_L$ XOR P_i

$X_R = F(X_L)$ XOR X_R

Swap X_L and X_R

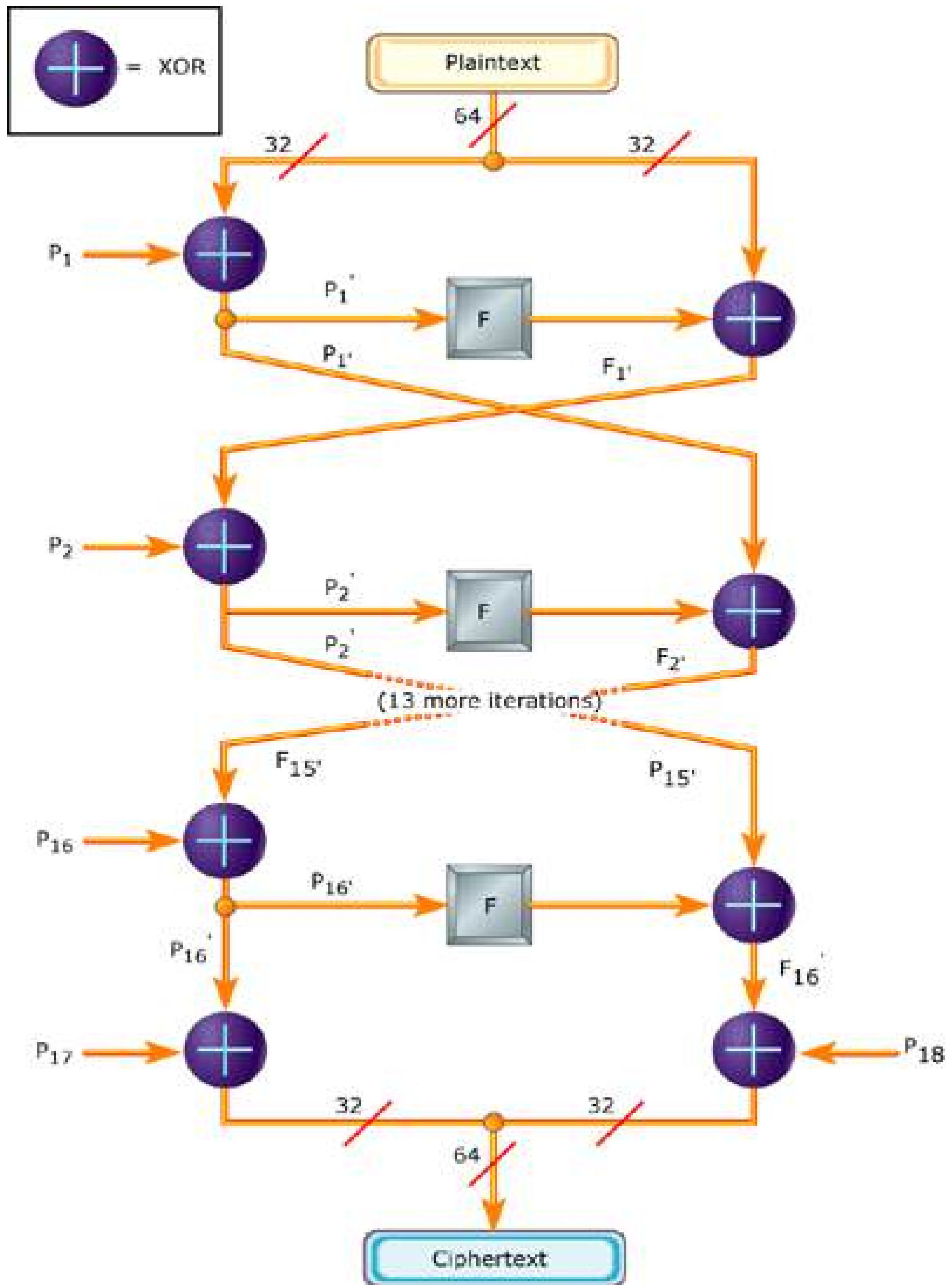
Swap X_L and X_R (Undo the last swap.)

$X_R = X_R$ XOR P_{17}

$X_L = X_L$ XOR P_{18}

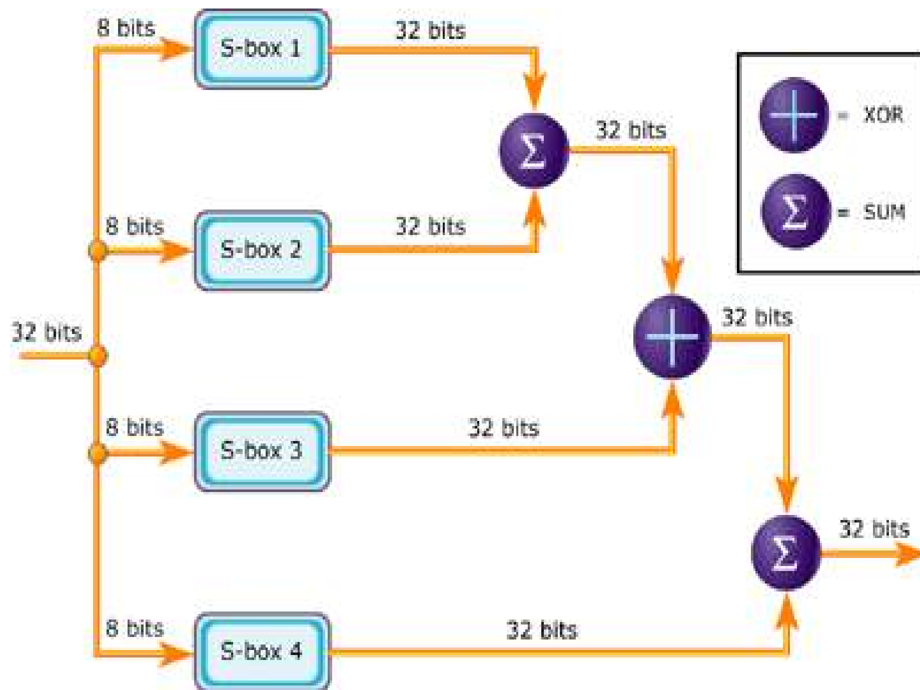
Recombine X_L and X_R

Prof.Dr.Hala Bahjat



(Flow chart of Blowfish algorithm)

FUNCTION F is as follows:



Divide X_L into four eight bit quarters :

a,b,c and d

$$F(X_L) = ((S_1[a] + S_2[b] \bmod 2^{32}) \text{ XOR } S_3[c]) + S_4[d] \bmod 2^{32})$$

- Decryption is exactly the same as encryption, except that P1, P2 ..... P18 are used in the reverse order.
- Implementations of Blowfish that require the fastest speeds should unroll the loop and ensure that all subkeys are stored in cache

Prof.Dr.

Blowfish facts

- Low key-agility and/or high memory demands makes Blowfish impractical in constrained environments.
- Small (64-bit) block size makes it insecure for applications that encrypt large amounts of data with the same key (such as data archival, file system encryption, etc.)
- Implemented in SSL and other security suites
- Blowfish's speed makes it an good choice for applications that encrypt intermediate amounts of data, such as typical of network communications (e-mail, file transfers).
- No attacks on Blowfish are known that work on the full 16-round official version (certain attacks recover some information from versions with up to 14-rounds).

5-Weakness of blowfish

Blowfish weak keys produce "bad" S-boxes, since Blowfish's S-boxes are key-dependent. There is a chosen plaintext attack against a reduced-round variant of Blowfish that is made easier by the use of weak keys. This is not a concern for full 16-round Blowfish. The sub key-generation algorithm does not assume that the key bits are random. Even highly correlated key bits, such as an alphanumeric ASCII string with the bit of every byte set to 0, will produce random sub keys. However, to produce sub keys with the same entropy, a longer alphanumeric key is required . The math behind Blowfish consists of 16 rounds, or loops. Cryptanalysis of Blowfish by Serge vaudenay reveals a partial differential attack that can recover the plaintext array in 28r+1 chosen plaintexts. There is also a class of known weak keys that can increase the effectiveness of this attack by a two facts.

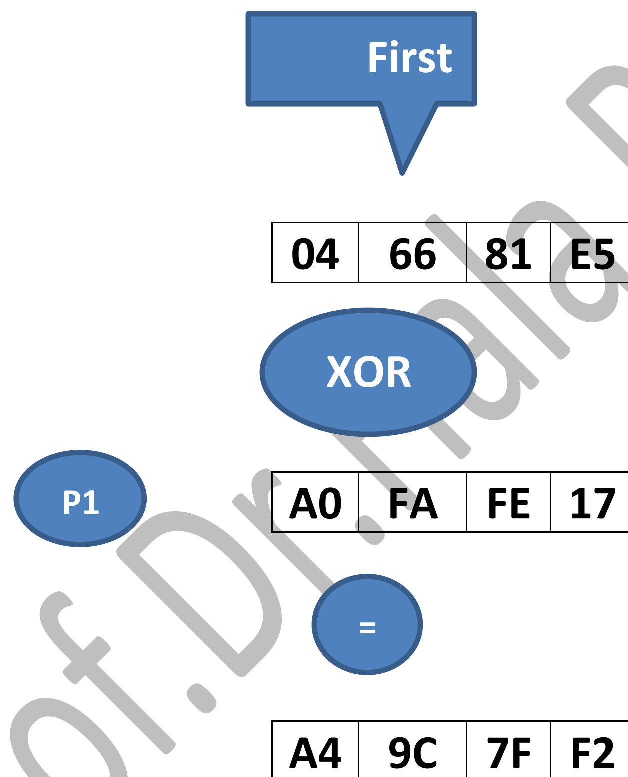
6-Encryption of blowfish whith example:

Prof.Dr.Hala Bahjat

1. We enter the plain text 64 bits and divide it in tow part the first part(XL) work XOR with p1.

Plain text

04	66	81	E5
AB	F7	15	88



2. Divide the new p1(32) to 4 part each part contain (8) bits each (8)bits enter to S-box and have been expands to become(32)bits.

.

Prof.Dr.Hala Bahjat

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(We do the operation between the output)

$$F(XL)=((S1,a+S2,b \bmod 2^{32}) \text{XOR } S3,c)+S4,d \bmod 2^{32}$$

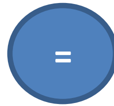
The result is :

88	54	2c	B1

3. We work XOR between result of the function and the other part of plain text (XR).

88	54	2c	B1

ab	F7	15	88



23	a3	39	39

Key scheduling:-

1- enter the plain text all with zero to generation p-array and 4(S-box).
2- The Key will enter by the user such as: (key=a0fafe1788542cb1…………………).
3- Enter the p-array randomly and also S-box.

P1 = 28aed2a6

P2= 85a308d3

P3 = 13198a2e

P4 = 03707344

.

.

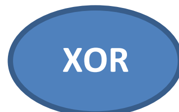
.

P18

4- work XOR between key and p-array.

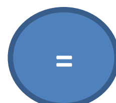
P1=

28	AE	D2	A6



Key=

A0	FA	FE	17



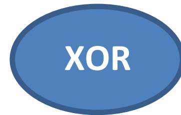
Prof.Dr.Hala Bahjat

P1(NEW)=

88	54	2C	B1

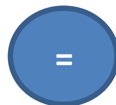
P2=

AB	F7	15	88



Key=

88	54	2C	B1



P2(NEW)=

23	A3	39	39

And we repeat the process for all p-array and key.

5- Enter the new p-array in the blowfish algorithm.
6- The result of algorithm is swap with p1 and p2.
7- The cipher text of algorithm is divide in tow part that will used the plain text to generate the (p-array) and used in encryption.

Prof.Dr.Hala Bahjat

Serpent

Serpent. It was developed by Ross Anderson (University of Cambridge Computer Laboratory), Eli Biham (Technion Israeli Institute of Technology), and Lars Knudsen (University of Bergen, Norway).

- Serpent encrypts a 128-bit plaintext *P* to a 128-bit cipher text *C*
- that a class of substitution-permutation networks (SPN)
- a 32-round
- a 256 bit external key
- is a symmetric block cipher

The Serpent Algorithm (encryption)

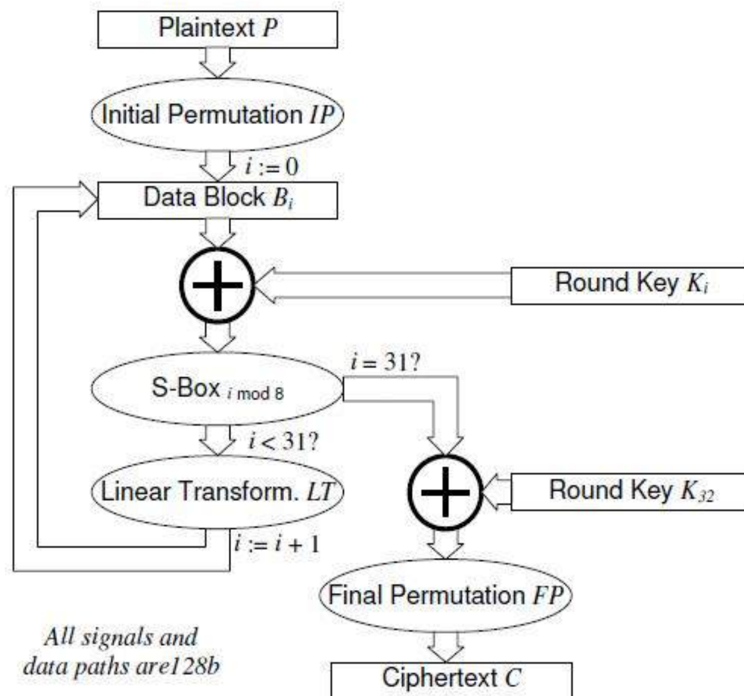
1-Initial permutation

2- Key Mixing: At each round, a 128-bit sub key *Ki* is exclusive or'ed with the current intermediate data *(Bi)*. (B_I)XOR(K_I)=32block,block(4bit)

3- S-Boxes: The 128-bit combination of input and key is considered as 4block,block(32bit). The S-box, which is implemented as a sequence of logical operations (as it would be in hardware) is applied to these four words.

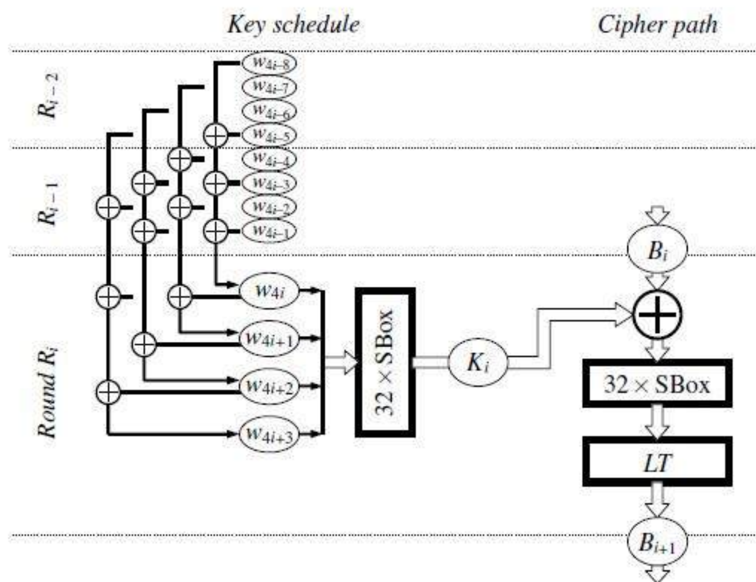
4- Linear Transformation: The 32 bits in each of the output words =128 bits are linearly mixed

Prof.Dr.Hala Bahjat



The Key Schedule

The user-supplied key is frst padded to 256 bits. This is done by assigning a 1 to the most significant bit, and a 0 to the remaining bits. The key is stored as eight 32-bit words w_{-8}.... w_{-1}. This is used to generate the prekeys (w_0, w_1..... w_{131}) with the recurrence:




```
wi=(wi-8⊕wi-5⊕wi-3⊕wi-1⊕φ⊕(i-8))⋘11 --------counting words8-15
```

```
wi=(wi-8⊕wi-5⊕wi-3⊕wi-1⊕φ⊕i)⋘11-----------counting word8-131
```

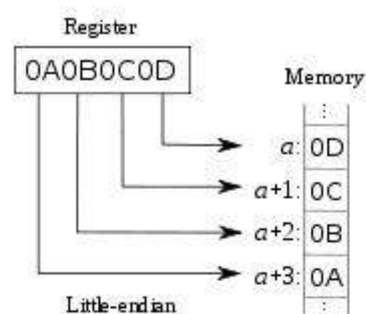
where φ is the hexadecimal constant (9e3779b9), <<<11 denote rotation

The round keys are now calculated from the prekeys using the S-boxes We use the S-boxes to transform the prekeys *wi* into words *ki* of round key in the following way:

$$
\begin{aligned}
\{k_0, k_1, k_2, k_3\} &:= S_3(w_0, w_1, w_2, w_3) \\
\{k_4, k_5, k_6, k_7\} &:= S_2(w_4, w_5, w_6, w_7) \\
\{k_8, k_9, k_{10}, k_{11}\} &:= S_1(w_8, w_9, w_{10}, w_{11}) \\
\{k_{12}, k_{13}, k_{14}, k_{15}\} &:= S_0(w_{12}, w_{13}, w_{14}, w_{15}) \\
\{k_{16}, k_{17}, k_{18}, k_{19}\} &:= S_7(w_{16}, w_{17}, w_{18}, w_{19}) \\
&\dots \\
\{k_{124}, k_{125}, k_{126}, k_{127}\} &:= S_4(w_{124}, w_{125}, w_{126}, w_{127}) \\
\{k_{128}, k_{129}, k_{130}, k_{131}\} &:= S_3(w_{128}, w_{129}, w_{130}, w_{131})
\end{aligned}
$$

☒ Little endian

The terms **endian** refer to the convention used to interpret the bytes making up a data word when those bytes are stored in computer memory.



Prof.Dr.Hala Bahjat